PENGUJIAN AKURASI AKOR PADA SUARA MUSIK PIANO DENGAN METODE CONVOLUTIONAL NEURAL NETWORK (CNN)

Oleh :

MUHAMMAD KAMAL YASYA

92216162

PROGRAM PASCA SARJANA
UNIVERSITAS GUNADARMA
JAKARTA
2020

PENGUJIAN AKURASI AKOR PADA SUARA MUSIK PIANO DENGAN METODE CONVOLUTIONAL NEURAL NETWORK (CNN)

Oleh :
MUHAMMAD KAMAL YASYA
92216162

TESIS

Untuk Memenuhi Salah Satu Syarat Guna Memperoleh

Gelar Magister Manajemen Sistem Informasi

Program Pasca Sarjana

Universitas Gunadarma

PROGRAM PASCA SARJANA
UNIVERSITAS GUNADARMA
JAKARTA
2020

Pernyataan Orisinalitas dan Publikasi

Saya yang bertanda tangan di bawah ini:

Nama	:	Muhammad Kamal Yasya
NIM	:	92216162
Judul Penelitian	:	Pengujian Akurasi Akor Pada Suara Musik Piano Dengan Metode Convolutional Neural Network (CNN)
Tanggal Sidang	:	-
Tanggal Lulus	:	-

Menyatakan bahwa tulisan di atas merupakan hasil karya saya sendiri dan dapat dipublikasikan sepenuhnya oleh Universitas Gunadarma. Segala kutipan dalam bentuk apapun telah mengikuti kaidah dan etika yang berlaku. Semua hak cipta dari logo serta produk yang disebut dalam buku ini adalah milik masing-masing pemegang haknya, kecuali disebutkan lain. Mengenai isi dan tulisan merupakan tanggung jawab Penulis, bukan Universitas Gunadarma.

Demikian pernyataan ini dibuat dengan sebenarnya dan dengan penuh kesadaran.

Jakarta, Maret 2020

Muhammad Kamal Yasya

Halaman Pengesahan

Judul Penelitian	:	Pengujian Akurasi Akor Pada Suara Musik Piano Dengan Metode Convolutional Neural Network (CNN)
Nama	:	Muhammad Kamal Yasya
NIM	:	92216162
Tanggal Lulus	:	-

Menyetujui,

Komisi Pembimbing

Dr. Onny Marlen, S.Kom., MMSI

(Ketua)

Dr. Hustinawati S.Kom., MMSI

(Ketua Program Studi)

Program Pasca Sarjana Magister Teknologi dan Rekayasa

Dr. Tubagus Maulana Kusuma, S.Kom., M.Eng.Sc.

(Direktur)

Abstraksi

Muhammad Kamal Yasya, 92216162
PENGUJIAN AKURASI AKOR PADA SUARA MUSIK PIANO DENGAN METODE CONVOLUTIONAL NEURAL NETWORK (CNN).

Kata Kunci : Convolutional Neural Network (CNN), Chromagram, Chord Recognition

(xviii+ 115+ lampiran)

Implementasi kecerdasan buatan dalam dunia diskusi telah diklasifikasikan banyak, tetapi banyak dari implementasi ini masih kurang akurat dalam hal akurasi seperti *Chord Recognition*. Pengenalan Akor adalah sistem atau aplikasi untuk mengetahui atau mengukur akor dengan benar. Untuk membangun sistem, diperlukan metode agar akurasi yang dihasilkan tinggi. Beberapa metode yang sering digunakan adalah metode *Convolutional Neural Network* (CNN). Penelitian ini bertujuan untuk mengukur akurasi pengenalan akor dari *piano* dan *audio bass* dengan menggunakan metode *Convolutional Neural Network* (*CNN*) dan memanfaatkan *chromagram* dalam transformasi *audio* ke bentuk gambar. Data yang digunakan dalam penelitian ini terdiri dari 12 kelas utama dan 12 kelas minor. Berdasarkan hasil tes yang dilakukan, akurasi setiap kelas memberikan nilai F1 yang cukup bagus. Nilai rata-rata F1 dari tes pengenalan akor keseluruhan berdasarkan nada akar adalah 87%.

Daftar Pustaka (2015 - 2020)

Abstract

Muhammad Kamal Yasya, 92216162
TESTING THE ACCURACY OF CHORDS IN PIANO MUSIC USING THE CONVOLUTIONAL NEURAL NETWORK (CNN) METHOD.

Keywords : Convolutional Neural Network (CNN), Chromagram, Chord Recognition

(xviii+ 115+ attachment)

The implementation of artificial intelligence in the world of discussion has been classified as many, but many of these implementations are still less accurate in terms of accuracy such as Chord Recognition. Chord Recognition is a system or application for knowing or measuring chords correctly. To build the system, methods are needed so that the accuracy generated is high. Some methods that are often used are Convolutional Neural Network (CNN) methods. This study aims to measure the accuracy of chord recognition from piano and bass audio by using the Convolutional Neural Network (CNN) method and utilizing chromagrams in the transformation of audio to image form. The data used in this study consisted of 12 major classes and 12 minor classes. Based on the results of tests conducted, the accuracy of each class gives a pretty good F1 value. The average F1 value of the whole chord recognition test based on the root tone is 87%.

Bibliography (2015 - 2020)

Daftar Riwayat Hidup

1. Identitas Diri

Nama	:	Muhammad Kamal Yasya
NIM	:	92216162
Tempat/Tanggal Lahir	:	Bekasi, 20 September 1994
Alamat Rumah	:	Royal Platinum Residence Blok B No 1-2 rt.001 rw.009 Kel. Mustika Jaya, Kec. Mustika Jaya Kota. Bekasi, Jawa Barat - 17510
Email	:	kamalyasya@gmail.com

2. Riwayat Pendidikan

Sarjana Sistem Informasi Universitas Gunadarma	2012-2016
SMAN 5 Tambun Selatan	2009-2012
SMPN 26 Kota Bekasi	2006-2009
SDN Lambang Sari 04	2000-2006

3. Pengalaman Kerja

Senior Test Engineer & Tech. Expert PT. Global Media Visual	Okt 2019-Sekarang
QA Manual & QA Automation PT. Prudential Life Assurance	Okt 2018-Okt 2019
QA Manual & Data Analis PT. Bank Mandiri	Jan 2017-Okt 2018
Java Developer PT. Digital Artha Media	Jul 2016-Des 2016
Asisten Tetap Laboratorium Sistem Informasi Universitas Gunadarma	Jul 2014-Juni 2016

Jakarta, Maret 2020

Muhammad Kamal Yasya, S. Kom

Kata Pengantar

Segala puji dan syukur penulis naikkan ke hadirat Allah SWT yang Maha Kuasa yang telah memberikan berkat, anugerah dan karunia yang melimpah, sehingga penulis dapat menyelesaikan tesis yang berjudul "PENGUJIAN AKURASI AKOR PADA SUARA MUSIK PIANO DENGAN METODE CONVOLUTIONAL NEURAL NETWORK (CNN)".

Pada kesempatan ini penulis menyampaikan terima kasih yang sebesar-besarnya kepada:

1. Ibu Prof. Dr. E.S. Margianti, SE, MM, selaku Rektor Universitas Gunadarma yang telah memberikan kesempatan kepada penulis untuk melanjutkan pendidikan dan dalam penyusunan tesis ini.

2. Bapak Prof. Suryadi Harmanto, SSi., MMSI, Selaku Pembantu Rektor II Universitas Gunadarma.

3. Bapak Dr. Tubagus Maulana Kusuma, S.Kom., M.Eng.Sc., selaku Direktur Program Pasca Sarjana Universitas Gunadarma.

4. Ibu Dr. Hustinawati S.Kom., MMSI, selaku Ketua Prodi Program Pasca Sarjana Universitas Gunadarma.

5. Bapak Dr. Onny Marlen, S.Kom., MMSI, selaku dosen pembimbing serta dosen pengajar yang telah banyak meluangkan waktu, tenaga serta pikiran dalam membimbing dan mengarahkan penulis untuk menyelesaikan penelitian tesis ini.

6. Ibu Dr. Bertalya, selaku dosen pendamping yang banyak membantu dalam mengarahkan penulis untuk menyelesaikan tesis ini.

7. Ayahanda serta ibunda yang telah banyak memberikan dorongan dan semangat tak ternilai serta doa restu mereka jugalah terlaksananya penulisan tesis ini.

8. Chris Muladi R, Panji Tamzil, Cincin Jati Sudarminto, Nadiailhaq N, Muhammad Sidik, Refha Febriana, Kasman Wicaksono, Roy Yohannes, Arvin yang telah mambantu untuk memberikan solusi dan kesempatan untuk konsultasi tesis ini.

9. Seluruh rekan-rekan angkatan 51 SIB seperjuangan di Universitas Gunadarma yang telah banyak membantu penulis.

10. Semua pihak yang tidak tersebutkan yang telah membantu penyelesaian tesis ini, penulis ucapkan juga terima kasih atas segala bantuan dan sarannya.

Sebagai manusia biasa yang tak luput dari kesalahan, maka penulis meminta maaf atas segala kekurangan dan keterbatasan dalam penyusunan tesis ini.

Akhir kata, hanya kepada Tuhan jualah segalanya dikembalikan dan penulis sadari bahwa penelitian ini masih jauh dari sempurna, disebabkan karena berbagai keterbatasan yang penulis miliki. Untuk itu penulis mengharapkan kritik dan saran yang bersifat membangun untuk menjadi perbaikan di masa yang akan datang.

Semoga apa yang ada pada penelitian tesis ini dapat bermanfaat bagi kita semua. Amin.

Jakarta, Maret 2020

Penulis

Daftar Isi

Daftar Gambar

Daftar Tabel

Daftar Lampiran

Bab 1

PENDAHULUAN

1.1 Latar Belakang Masalah

Seiring dengan perkembangan teknologi yang semakin cepat, kecerdasan yang awalnya hanya dimiliki oleh makhluk hidup, sekarang mulai dikembangkan dan diimplementasikan ke dalam komputer. Hal tersebut biasa disebut dengan Kecerdasan Buatan. Kecerdasan Buatan atau dalam bahasa Inggris disebut *Artificial Intelligence* (AI) didefinisikan sebagai salah satu cabang ilmu pengetahuan yang berhubungan dengan pemanfaatan mesin untuk memecahkan persoalan yang rumit dengan cara yang lebih manusiawi. Teknologi kecerdasan buatan memiliki banyak kegunaan sehingga mampu membuat komputer dan perangkat selular untuk melakukan verifikasi wajah, pengenalan suara, pengenalan sidik jari, memberikan rekomendasi *video*, dan masih banyak lainnya. Selain diimplementasikan di ranah teknologi, kecerdasan buatan juga telah diimplementasikan di ranah lainnya, seperti kesehatan, otomotif, hingga permusikan.

Namun, meskipun implementasi kecerdasan buatan dalam dunia permusikan tergolong banyak, namun dalam hal akurasi masih kurang akurat seperti *Chord Recognition*. *Chord Recognition* adalah sebuah sistem atau aplikasi untuk mengetahui atau mengukur akor secara tepat. Untuk membangun sistem tersebut, dibutuhkan metode-metode agar akurasi yang dihasilkan tinggi. Metode yang sering digunakan adalah metode *Convolutional Neural Network* (CNN).

Convolutional Neural Network (CNN) adalah salah satu metode *Machine Learning* dari pengembangan *Multi Layer Perceptron* (MLP) yang didesain untuk mengolah data dua dimensi. CNN dikatakan pengembangan lebih lanjut dari MLP karena CNN menggunakan metode yang mirip dengan MLP, namun menggunakan dimensi yang lebih banyak. CNN juga sering digunakan sebagai ekstraktor fitur yang kuat dari data yang telah disatukan, seperti gambar. Metode ini dapat diperluas ke berbagai tugas klasifikasi sinyal audio dengan merepresentasikan sinyal input dalam domain frekuensi waktu.

Terdapat beberapa peneliti terdahulu yang telah melakukan penelitian serupa, seperti Filip Korzeniowski dan Gerhard Widmer telah melakukan penelitian tentang pengenalan akor. Penelitian tersebut berjudul FEATURE LEARNING FOR CHORD RECOGNITION: THE DEEP CHROMA EXTRACTOR. Pada penelitian tersebut, Filip dan Gerhard melakukan pelatihan data menggunakan *chromogram* terhadap *Short-Time Fourier Transform* (STFT) sebagai data latih. STFT merupakan sebuah metode yang mengubah data mentah audio ke bentuk satuan signal menjadi frekuensi dalam satuan waktu. Penelitian tersebut juga membandingkan hasil akurasi *chromogram* dengan *spectogram*. Tingkat akurasi yang didapat menggunakan *chromogram* yaitu sebesar 69.2%, sementara tingkat akurasi untuk model yang menggunakan *spectogram* adalah sebesar 78.8%.

Beberapa studi penelitian menggunakan metode CNN pada model arsitektur untuk melakukan klasifikasi terhadap data berupa suara. Penelitian tersebut menerapkan CNN asli yang diperluas secara fungsional untuk *input spectogram* suara dan menunjukkan bahwa arsitektur CNN mengungguli bentuk-bentuk dasar sebelumnya dari *Deep Learning Network* (DNN) yang terhubung penuh pada pengenalan telepon dan tugas-tugas besar pengenalan suara vokal. Adapun sebuah penelitian untuk melakukan klasifikasi kelaparan, kesakitan dan mengantuk pada suara bayi menangis menggunakan CNN terhadap *spectogram* MFCC dengan tingkat akurasi 78.5%.

Beberapa studi penelitian lainnya, seperti yang dilakukan oleh I Gede Harsemadi, Made Sudarman, dan Nyoman Pramaita yakni memanfaatkan algoritma KNN dalam mengelompokkan musik terhadap suasana hati. Sistem yang dibangun dalam penelitian tersebut akan menerima masukan data berupa *file* musik format mono *.wav, yang selanjutnya melakukan proses pengelompokan terhadap musik dengan menggunakan klasifikasi KNN.

Sistem tersebut akan menghasilkan keluaran berupa label jenis *mood* yaitu, *contentment*/kepuasan, *exuberance*/gembira, *depression*/depresi dan *anxious*/cemas; kalut. Secara umum hasil akurasi sistem dengan menggunakan algoritma klasifikasi K-NN cukup baik yaitu 86,55% pada nilai k = 3, serta waktu pemrosesan klasifikasi rata-rata 0,01021 detik per-file musik.

1.2 Identifikasi Masalah

Berdasarkan latar belakang telah diuraikan, maka penelitian ini akan menguji akurasi metode CNN dalam melakukan pengenalan terhadap data lagu yang dimainkan dengan menggunakan alat musik Piano. Data latih yang digunakan adalah suara instrumen piano dalam bentuk format mp3 dan wav. Data latih akan direkam dengan bantuan aplikasi Audacity dan akan dipisah menjadi potongan-potongan lagu berdasarkan jenis akor dan not pada piano.

1.3 Batasan Masalah

Hasil penelitian ini memiliki implikasi baik untuk penelitian dan pengembangan lebih lanjut. Agar pengerjaan lebih terarah, dalam penelitian ini terdapat beberapa batasan masalah, diantaranya :

- Jumlah instrumen yang digunakan adalah instrumen piano.
- Format file instrumen yang dapat diterima hanya file dengan format mp3 dan wav.
- Data yang diuji terdiri dari 12 kelas Major dan 12 kelas Minor.

1.4 Tujuan Penelitian

Adapun tujuan dari penelitian ini adalah untuk mengukur tingkat akurasi dari metode *Convolutional Neural Network* (CNN) dalam melakukan pengenalan terhadap data lagu yang dimainkan dengan menggunakan alat musik Piano.

1.5 Kegunaan Penelitian

Penelitian ini memiliki beberapa kegunaan yaitu sebagai berikut :

1.5.1 Kegunaan Teoritis

Kegunaan Teoritis pada penelitian ini di bagi menjadi dua, yaitu kegunaan penelitian untuk peneliti, dan kegunaan penelitian untuk peneliti selanjutnya.

1.5.1.1 Peneliti

Penelitian ini dapat digunakan sebagai tambahan pengetahuan tentang metode CNN dalam melakukan pengenalan otomatis. Selain itu, juga menambah pengetahuan peneliti tentang bahasa pemrograman Python.

1.5.1.2 Peneliti Selanjutnya

Penelitian ini dapat digunakan untuk sebagai referensi untuk penelitian selanjutnya bagi peneliti lain yang berminat untuk mempelajari Kecerdasan Buatan. Selain itu, penelitian ini juga dapat digunakan untuk mengimplementasikan metode *Machine Learning*.

1.5.2 Kegunaan Praktis

Kegunaan praktis pada penelitian ini adalah untuk pemusik amatir.

1.5.2.1 Pemusik Amatir

Penelitian ini dapat digunakan sebagai bahan acuan untuk menentukan akor dari sebuah lagu atau nada. Selain itu, penelitian ini juga sangat membantu para pemusik amatir yang masih buta dengan not nada.

Bab 2

TELAAH PUSTAKA

2.1 Tinjauan Pustaka Penelitian

Penelitian ini dilakukan dengan terlebih dahulu membahas sejumlah teori dan konsep yang berhubungan dengan metode CNN. Landasan teori yang dibahas terdiri dari audio, akor, *bass*, *Beat Tracking*, *Chromagram*, dan Metode CNN.

2.1.1 Audio

Audio adalah sebuah deret waktu, dimana sumbu y adalah amplitudo arus yang sesuai dengan sebuah membran loudspeaker dan sumbu x adalah sumbu yang sesuai dengan satuan waktu data tersebut. Telinga manusia hanya dapat mendengar bunyi dengan rentang frekuensi antara 20 Hz hingga 20 KHz (20.000Hz). Angka 20 Hz sebagai frekuensi suara terendah yang dapat didengar, sedangkan 20 KHz merupakan frekuensi tertinggi yang dapat didengar. Gelombang suara mengandung sejumlah komponen penting, seperti amplitudo, panjang gelombang, dan frekuensi. Komponen-komponen tersebut mampu membuat suara yang satu berbeda dengan suara yang lain.

Amplitudo sendiri merupakan kekuatan atau daya gelombang sebuah sinyal. Nilai amplitudo diinterpretasikan sebagai volume. Semakin besar nilai dari sebuah amplitudo, maka semakin keras suara yang dihasilkan. Sebaliknya, jika nilai dari suatu amplitudo semakin kecil, maka suara yang dihasilkan semakin lemah. Frekuensi adalah jumlah dari siklus yang terjadi dalam satu detik dan memiliki satuan Hertz (Hz). Getaran gelombang suara yang cepat membuat frekuensi semakin tinggi. Misalnya, menyanyi nada tinggi membuat tali suara pada pita suara bergetar secara cepat.

2.1.2 *Akor*

Akor adalah sebuah kombinasi tiga nada atau lebih yang dibunyikan secara bersamaan. Akor tiga not yang berjarak tiga scale pada root sampai not ketiga disebut triad. Untuk mempermudah pemusik dalam menentukan apakah suatu kombinasi dari tiga nada tertentu merupakan sebuah triad atau bukan, maka pemusik dapat membentuk diagram circle of thirds. Scale atau skala nada merupakan jarak yang menandakan lokasi suatu nada dengan titik mulai tertentu. Terdapat empat jenis triad yang dibedakan berdasarkan skala nadanya, diantaranya :

1. *Major Triad*

   Major triad merupakan tiga nada yang dibunyikan dengan skala nada 1-2- 1,5. Akor major menunjukan identitas suatu bagian lagu lebih dekat dengan root karena ada satu buah akor major yang menempati Scale I.

2. *Minor triad*

   Minor triad merupakan tiga nada yang dibunyikan dengan skala 1-1,5-2.

3. *Diminished triad*

   Diminished triad merupakan tiga nada yang dibunyikan dengan skala 1- 1,5-1,5.

4. *Augmented triad*

   Augmented triad merupakan tiga nada yang dibunyikan dengan skala 1- 1,5-2,5.

Akor *seventh* merupakan akor empat not yang kelas *pitch*nya dapat diatur sebagai sepertiga. Skala nada akor major *seventh* adalah 1-2-1,5-2. Sedangkan skala nada untuk minor *seventh* adalah 1-1,5-2-2. Sama halnya dengan triad, kelas nada yang dimiliki oleh akor *seventh* menempati posisi yang berdekatan (rumpun empat kelas) pada lingkaran pertiga. Keempat anggota akor ketujuh adalah *root*, *third*, *fifth*, dan *seventh*. Terdapat suatu cara untuk melakukan teknik *blocking* pada akor minor *seventh*. Dengan memanfaatkan teknik *blocking*, maka *root* dapat berperan sebagai nada *bass*. Sementara *third*, *fifth*, dan *seventh* sebagai major triad.

2.1.3 *Beat Tracking*

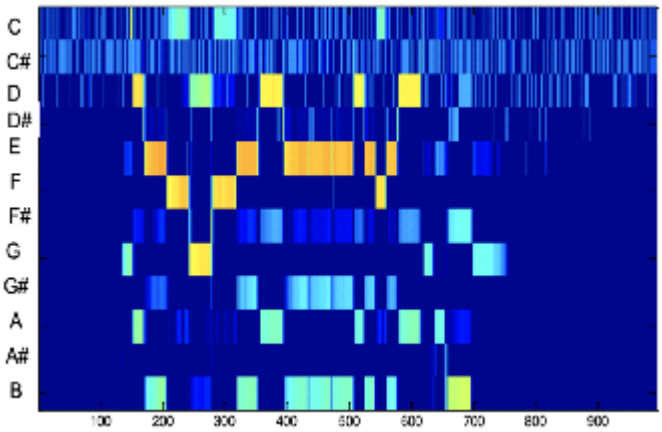
Beat tracking digunakan untuk menentukan contoh waktu dalam rekaman audio, di mana pendengar manusia cenderung mengetuk kakinya ke musik. *Beat tracking* pada rancangan menggunakan metode *dynamic programming*. *Dynamic programming* atau *dynamic optimization* bukan merupakan rumus matematika yang mampu memberikan jawaban dengan hanya sekedar memberikan input. Sebaliknya, *dynamic programming* adalah kombinasi pemikiran terstruktur dan pola pikir analitis untuk menyelesaikan sebuah permasalahan.

Untuk melakukan *beat tracking*, hal yang harus dilakukan terlebih dahulu adalah mencari sinyal yang berupa onset. Onset merupakan lokasi dimana sinyal berada pada nilai yang tinggi secara tiba-tiba. Onset yang ditemukan kemudian dilakukan penandaan. Onset yang ditemukan bisa saja merupakan *false positive*. Untuk mengurangi *false positive* tersebut, maka harus dilakukan perhitungan untuk menemukan urutan umum terpanjang dari onset.

2.1.4 Chromagram

Chromagram adalah transformasi properti frekuensi-waktu sinyal menjadi prekursor *pitch* yang berubah-ubah untuk sementara waktu. Transformasi ini didasarkan pada pengamatan persepsi tentang sistem pendengaran dan telah terbukti memiliki beberapa sifat matematika yang menarik. *Chromagram* memperluas konsep chroma untuk memasukkan dimensi waktu. Seperti halnya kita menggunakan *spektrogram* untuk menyimpulkan properti tentang distribusi energi sinyal dari frekuensi dan waktu, *chromagram* dapat digunakan untuk menyimpulkan properti tentang distribusi energi sinyal terhadap kroma dan waktu.

Terdapat dua masalah yang muncul dalam mengembangkan konsep *chromagram* ini, yakni definisi dan bagaimana cara menghitung *chromagram*. *Chroma* adalah pemetaan banyak-ke-satu (*many-to-one*) frekuensi, sementara *chromagram* adalah ukuran kekuatan sinyal sebagai fungsi dari *chroma* dan waktu. Oleh karena itu, *chromagram* dapat didefinisikan sebagai pemetaan banyak-ke-satu (*many-to-one*) kekuatan sinyal pada frekuensi milik kelas *chroma* yang sama. Pembagian antara transformasi langsung dari sinyal asli dan transformasi dari gambar frekuensi waktu dapat digunakan untuk menghitung *chromagram*. Contoh *chromagram* dapat dilihat dengan jelas pada Gambar 2.1.



Gambar 2.1: Chromagram dari sebuah alat musik Clarinet

2.1.5 *Windowing*

Windowing berfungsi untuk meminimalisir sinyal yang tak kontinu pada awal dan akhir masing-masing *frame*. Nilai panjang filter didapatkan dengan menggunakan rumus di bawah. Nilai panjang tersebut akan dipakai untuk membuat respon *impulse* di simulasi dan implementasi. Nilai panjang *filter* juga akan berpengaruh kepada faktor *roll-off* atau faktor kelandaian dari suatu *filter*.

Rumus untuk menghitung *window* dapat dilihat pada persamaan

1.

$$Yt(n) = xt(n)w(n), 0 \leq n \leq N-1 \text{ (1)}$$

Rumus untuk menghitung *Hamming Window* dapat dilihat pada persamaan

2.

$$w(n) = 0.54 - 0.46coscos(2N\pi - n1), 0 \leq n \leq N-1 \text{(2)}$$

Keterangan:

N = Panjang setiap frame

$xt(n)$ = Sinyal masukan ke-n pada frame ke-t

$w(n)$ = Fungsi Hamming Window

$Yt(n)$ = Nilai hasil windowing untuk sinyal masukan ke-n pada frame ke-t

Data mentah dari masukan yang pertama kali dibaca oleh bahasa pemrograman Python adalah sinyal dalam domain waktu. Data sinyal dalam domain waktu tersebut perlu diubah ke dalam satuan frekuensi. Proses untuk mengubah data mentah yang berbentuk sinyal tersebut diubah dengan menggunakan metode *Discrete Fourier Transform* (DFT). Pada tahap ini, setiap *frame* yang terdiri dari N sampel dikonversi dari domain waktu ke domain frekuensi. Rumus dari DFT dapat dilihat di dalam persamaan 3.

$$X(k) = \sum Nn = -01x(n)e(-j2N\pi nk), k = 0, 1, ..., N - 1 \quad (3)$$

Keterangan:
$X(k)$ = Nilai transformasi Fourier (FT) ke-k
$x(n)$ = Input signal ke-n
N = Panjang input signal

Hasil proses windowing pada persamaan 1 kemudian dimasukan sebagai input signal ke-n, sehingga menghasilkan persamaan 4.

$$Zt(k) = \sum Nn = -01Yt(n)e(j2N\pi nk), k = 0, 1, ..., N - 1 \quad (4)$$

$$ei\theta = cos\ cos\theta + sin\ sin\theta \quad (5)$$

Berdasarkan rumus Euler pada persamaan 5, dengan e adalah bilangan eksponen, i adalah bilangan imajiner, dan cos dengan sin adalah fungsi trigonometri. Jika persamaan 5 disubstitusikan ke persamaan 4 maka rumus DFT menjadi persamaan 6 di bawah ini.

$$Zt(k) = \sum Nn = -01Yt(n)[cos\ cos(2\pi nkN) - jsinsin(2\pi nkN)] \quad (6)$$

k = 0,1, ... , $N - 1$
Keterangan:
$Zt(k)$ = Nilai FT yang mengandung bilangan kompoleks
$Yt(n)$ = Nilai hamming window ke-n pada frame ke-t
j = Bilangan imajiner

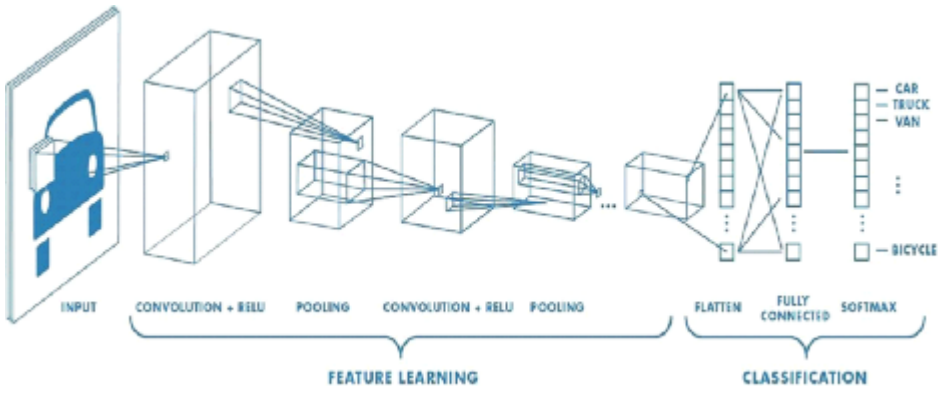
Hasil perhitungan pada persamaan 6 mengandung bilangan riil dan imajiner. Oleh karena itu, nilai Fourier spektrum didapat dari nilai *magnitude* yang dibentuk kedua bilangan riil dan imajiner. Besar nilai *magnitude* dari persamaan z = x + y tersebut dihitung di dalam persamaan 7.

$$r = M(n) = \sqrt{x2 + y2} \quad (7)$$

Keterangan:
z = Nilai FT yang mengandung bilangan kompleks
$M(n)$ = Magnitude
x = koefisien bagian riil
y = Koefisien bagian imajiner

2.1.6 *Convolutional Neural Network (CNN)*

Convolutional Neural Network (CNN) adalah salah satu jenis *neural network* yang biasa digunakan pada data *image*. CNN bisa digunakan untuk mendeteksi dan mengenali objek pada sebuah *image*. CNN terdiri dari *neuron* yang memiliki *weight*, bias, dan fungsi aktivasi. Cara kerja CNN memiliki kesamaan pada MLP, namun dalam CNN setiap *neuron* dipresentasikan dalam bentuk dua dimensi, tidak seperti MLP yang setiap *neuron* hanya berukuran satu dimensi. Pada CNN, data yang dipropagasikan oleh jaringan adalah data dua dimensi, sehingga operasi linear dan parameter bobot pada CNN berbeda. Pada CNN operasi linear menggunakan operasi konvolusi, sedangkan bobot tidak lagi satu dimensi saja, namun berbentuk empat dimensi yang merupakan kumpulan kernel konvolusi. Karena sifat proses konvolusi, maka CNN hanya dapat digunakan pada data yang memiliki struktur dua dimensi seperti citra dan suara. Arsitektur dari CNN dibagi menjadi dua bagian besar, *Feature Extraction Layer* dan *Fully-Connected Layer* (MLP). Arsitektur CNN dapat dilihat pada Gambar 2.2.



Gambar 2.2: Arsitektur *Convolutional Neural Network*

I Wayan Suartika Eka Putra (2016) dalam penelitiannya yang berjudul Klasifikasi Citra Menggunakan *Convolutional Neural Network* (CNN) pada *Caltech 101* mengimplementasikan. salah satu metode *machine learning* yang dapat digunakan untuk klasifikasi citra objek yaitu CNN.

Pada penelitian yang dilakukan, peneliti mengimplementasikan metode CNN yang terdiri dari dua tahap. Tahap pertama adalah klasifikasi citra menggunakan *feedforward*. Tahap kedua merupakan tahap pembelajaran dengan metode *backpropagation*. Hasil uji coba dari klasifikasi citra objek dengan tingkat *confusion* yang berbeda pada basis data *Caltech 101* menghasilkan rata-rata nilai akurasi tergolong tinggi, sehingga dapat disimpulkan bahwa metode CNN yang digunakan pada penelitian tersebut mampu melakukan klasifikasi dengan baik.

Berbeda dengan Dzikry Maulana Hakim dan Ednawati Rainarli(2019) dalam penelitiannya yang berjudul *Convolutional Neural Network* untuk Pengenalan Citra Notasi Musik. Peneliti melakukan pengujian untuk melakukan pengenalan citra notasi musik dengan dua cara. Pertama, peneliti menguji performa CNN menggunakan data notasi musik yang telah dipotong dan yang kedua peneliti melakukan pengujian menggunakan sebaris notasi musik. Nilai akurasi yang didapatkan untuk pengenalan sebaris notasi musik tidak terlalu besar, yaitu 26,19%. Walaupun untuk proses segmentasi masih belum maksimal dalam memotong setiap notasi, namun metode CNN bekerja sangat baik untuk mengenali setiap notasi musik yang telah dipotong dengan benar. Hal ini ditunjukkan dari nilai akurasi yang mencapai 95,56%.

2.1.7 *Music Information Retrieval*

Music Information Retrieval (MIR) terutama berkaitan dengan ekstraksi dan inferensi fitur yang berarti dari musik seperti sinyal *audio*, representasi simbolik atau sumber eksternal seperti halaman web. Berikut merupakan contoh tugas MIR, antara lain *fingerprinting*, *cover song detection*, *genre recognition*, *key detection*, *beat tracking*, *symbolic melodic similarity*, *source separation*, *pitch tracking*, *tempo estimation*, dan masih banyak lagi. MIR digunakan pada sistem untuk menyiapkan data mentah menjadi bentuk data yang dapat dilakukan observasi oleh model. Berikut merupakan beberapa fitur pada MIR yang digunakan oleh sistem :

1. *Beat Tracking*

   Sistem memanfaatkan fitur *beat tracking* yang disediakan oleh pustaka librosa yang menghasilkan keluaran dalam bentuk list. Nilai yang dihasilkan dalam satuan detik.

2. *Harmonic Separation*

   Fitur *harmonic separation* berfungsi untuk memisahkan data masukan berupa *harmonic* dan *percussive*. Contoh dari *harmonic* adalah nada yang dibunyikan oleh piano dan suara vokal, sedangkan *percussive* merupakan kumpulan data suara yang dimainkan dengan cara dipukul, digesek, maupun dikocok.

2.2 Penelitian Terdahulu

I Wayan Suartika Eka Putra (2016) dalam penelitiannya yang berjudul Klasifikasi Citra Menggunakan *Convolutional Neural Network* (CNN) pada *Caltech 101* mengimplementasikan salah satu metode *machine learning* yang dapat digunakan untuk klasifikasi citra objek yaitu CNN. Pada penelitian yang dilakukan, peneliti mengimplementasikan metode CNN yang terdiri dari dua tahap. Tahap pertama adalah klasifikasi citra menggunakan *feedforward*. Tahap kedua merupakan tahap pembelajaran dengan metode *backpropagation*. Sebelum melakukan klasifikasi, terlebih dahulu dilakukan praproses dengan metode *wrapping* dan *cropping* untuk memfokuskan objek yang akan diklasifikasi. Selanjutnya dilakukan *training* menggunakan metode *feedforward* dan *backpropagation*. Terakhir adalah tahap klasifikasi menggunakan metode *feedforward* dengan bobot dan bias yang diperbarui. Hasil uji coba dari klasifikasi citra objek dengan tingkat *confusion* yang berbeda pada basis data *Caltech 101* menghasilkan rata-rata nilai akurasi tergolong tinggi, sehingga dapat disimpulkan bahwa metode CNN yang digunakan pada penelitian tersebut mampu melakukan klasifikasi dengan baik.

Danny Lionel, Rudy Adipranata dan Endang Setyati (2016) dalam penelitian yang berjudul Klasifikasi Genre Musik Menggunakan Metode *Deep Learning Convolutional Neural Network* dan *MelSpektrogram* mengimplementasikan metode *Mel-spectrogram*. Dimana *Mel spektrogram* merupakan hasil pemetaan fitur yang telah diambil oleh metode MFCC, yang akan diklasifikasikan dan dimasukkan ke dalan *Convolutional Neural Network*. Yang akan dibedakan *activation function* nya yaitu ReLU dan ELU. Penelitian ini menunjukkan bahwa pengambilan fitur dari *audio* dengan menggunakan MFCC merupakan metode yang benar dan dalam hasil pengujian, banyaknya dataset, iterasi *training*, dan spesifikasi komputer sangat mempengaruhi tingkat akurasi dan lama pembuatan *neural network model* yang optimal. Dalam hasil penelitian ini telah diuji beberapa kali didapatkan hasil akurasi yang paling

optimal yaitu 99%.

Dzikry Maulana Hakim dan Ednawati Rainarli (2019) pada penelitiannya yang berjudul *Convolutional Neural Network* untuk Pengenalan Citra Notasi Musik melakukan penelitian untuk mengenali citra notasi musik dengan metode CNN. Pada penelitian ini, untuk pengenalan notasi musik digunakan *Convolutional Neural Network* (CNN). Arsitektur CNN yang dipakai adalah *kernel* 3x3, jumlah layer pada *feature learning* sebanyak 3 *convolutional layer*, 3 *pooling layer*, *filter* pada *convolutional layer* 64,128, 256, dan jumlah *neuron* pada *hidden layer* sebanyak 7168. Pengujian dilakukan dengan dua cara, yang pertama menguji performa CNN menggunakan data notasi musik yang telah dipotong dan yang kedua adalah melakukan pengujian menggunakan sebaris notasi musik. Nilai akurasi yang didapatkan untuk pengenalan sebaris notasi musik tidak terlalu besar, yaitu 26,19%. Walaupun untuk proses segmentasi masih belum maksimal dalam memotong setiap notasi, namun metode CNN bekerja sangat baik untuk mengenali setiap notasi musik yang telah dipotong dengan benar. Hal ini ditunjukkan dari nilai akurasi yang mencapai 95,56%.

Nick Collins melakukan penelitian dengan membuat *library SuperCollider Music Information Retrieval* (SCMIR). SCMIR adalah set ekstensi untuk bahasa pemrograman *audio* yang berfungsi untuk melakukan analisis otomatis sebuah *file audio*. *Library* ini mendukung teknologi pencarian informasi musik umum, termasuk untuk pemrosesan *batch* di seluruh *file* suara. *Library* memiliki kelebihan yang diambil dari mode *Non-Real-time scsynth* dan *plug-in* untuk ekstraksi fitur cepat, serta fitur tambahan untuk perhitungan intensif. Hasil dari penelitian ini adalah Nick Collins dapat meningkatkan hubungan antara sintesis pencocokan fitur, mendengarkan mesin dalam sistem interaktif, dan tugas MIR, melintasi *real time task* dan *non-real time* dengan memprioritaskan adopsi teknologi MIR dalam sistem interaktif. Selain itu, library SCMIR juga membawa teknologi pencarian informasi musik ke bahasa pemrograman komputer musik yang sudah terkenal. Rincian mengenai penelitian terdahulu dapat dilihat pada tabel 2.1.

Tabel 2.1: Penelitian Terdahulu.

No	Peneliti	Judul Penelitian	Metode	Hasil
1	I Wayan Suartika Eka Putra	Klasifikasi Citra Menggunakan Convolutional Neural Network (CNN) pada Caltech 101	Mengimplementasikan salah satu metode machine learning yang dapat digunakan untuk klasifikasi citra objek yaitu CNN	Hasil uji coba dari klasifikasi citra objek dengan tingkat confusion yang berbeda pada basis data Caltech 101 menghasilkan rata-rata nilai akurasi tergolong tinggi, sehingga dapat disimpulkan bahwa metode CNN yang digunakan pada penelitian tersebut mampu melakukan klasifikasi dengan baik
2	Danny Lionel, Rudy Adipranata dan Endang Setyati	Klasifikasi Genre Musik Menggunakan Metode Deep Learning Convolutional Neural Network dan MelSpektrogram	Mengimplementasikan metode agar suatu audio file dapat dikenali secara otomatis dari fiturfitur yang telah diekstrak sebelumnya dengan bantuan MFCC (Mel Frequency Cepstral Coefficients) dan ANN (Artificial Neural Network).	Penelitian ini menunjukkan bahwa pengambilan fitur dari audio dengan menggunakan MFCC merupakan metode yang benar dan dalam hasil pengujian, banyaknya dataset, iterasi training, dan spesifikasi komputer sangat mempengaruhi tingkat akurasi dan lama pembuatan neural network model yang optimal. Dalam hasil penelitian ini telah diuji beberapa kali didapatkan hasil akurasi yang paling optimal yaitu 99%

Tabel 2.1: Penelitian Terdahulu. (Lanjutan)

No	Peneliti	Judul Penelitian	Metode	Hasil
3	Dzikry Maulana Hakim dan Ednawati Rainarli	Convolutional Neural Network untuk Pengenalan Citra Notasi Musik	Mengetahui cara agar sistem dapat mendeteksi sebuah notasi musik dan kemudian mengenali notasi musik tersebut	Pengujian dilakukan dengan dua cara, yang pertama menguji performa CNN menggunakan data notasi musik yang telah dipotong dan yang kedua adalah melakukan pengujian menggunakan sebaris notasi musik. Nilai akurasi yang didapatkan untuk pengenalan sebaris notasi musik tidak terlalu besar, yaitu 26,19%. Walaupun untuk proses segmentasi masih belum maksimal dalam memotong setiap notasi, namun metode CNN bekerja sangat baik untuk mengenali setiap notasi musik yang telah dipotong dengan benar. Hal ini ditunjukkan dari nilai akurasi yang mencapai 95,56%.

Tabel 2.1: Penelitian Terdahulu. (Lanjutan)

4	Nick Collins	SCMIR: A SUPERCO-LLIDE R MUSIC IN-FORMATION RETRIEVAL LIBRARY	Membuat sebuah library extension set untuk menganalisis file audio secara otomatis	Peneliti berhasil mengembangkan library SCMIR sehingga dapat meningkatkan hubungan antara sintesis pencocokan fitur, mendengarkan mesin dalam sistem interaktif, dan mampu melintasi pekerjaan realtime dan non-realtime dengan memprioritaskan adopsi teknologi MIR dalam sistem interaktif.

Tabel 2.1: Penelitian Terdahulu. (Lanjutan)

5	Florian Krebs, Sebastian Bo¨ck, and Gerhard Widmer	RHYTHMIC PATTERN MODELING FOR BEAT AND DOWNBEAT TRACKING IN MUSICAL AUDIO	Menyelidiki penggunaan pemodelan pola ritmis untuk menyimpulkan struktur metrik dalam rekaman audio musikal	Penelitian yang dilakukan menunjukkan bahwa menghitung fitur onset setidaknya untuk dua band frekuensi yang berbeda meningkatkan kinerja pelacakan suram secara signifikan dibandingkan dengan fitur tunggal yang mencakup seluruh rentang frekuensi. Dalam perbandingan dengan enam sistem referensi, pemodelan gaya tarian secara eksplisit sebagai pola ritmik terbukti mengurangi kesalahan oktaf (mendeteksi setengah atau tempo ganda) dalam pelacakan ketukan. Selain itu, pelacakan suram ditingkatkan secara substansial dibandingkan dengan varian yang hanya memodelkan meter dan dua sistem referensi.

Tabel 2.1: Penelitian Terdahulu. (Lanjutan)

6	Albert Parlys, Ajub Ajulian Zahra, dan Achmad Hidayatno	Penggolongan Lagu Berdasarkan Spektogram dengan Convolutional Neural Network	Penelitian ini akan membahas perancangan sebuah sistem untuk menggolongan lagu berdasarkan spektogram. Masukan sistem berupa lagu dengan format audio MP3 yang diubah ke dalam bentuk spektogram kemudian dilatih menggunakan Convolutional Neural Network. Ciri lagu akan diperoleh kemudian diklasifikan ke dalam lima genre berbeda yaitu pop, rock, classic, dubstep, dan reggae	Berdasarkan hasil pelatihan dan pengujian dengan filter 3x3 didapat nilai akurasi penggolongan lagu sebesar 100% pada 750 data latih dan 98% pada 50 lagu data uji. Algoritme pembelajaran terbaik pada pelatihan dengan filter yang sama adalah algoritme Adam yang lebih cepat dibandingkan dengan Adadelta, Adagrad, dan SGD

Bab 3

METODE PENELITIAN

3.1 Metode Penelitian

Dalam melakukan penelitian ini, tahapan pertama yang dilakukan adalah pengumpulan data. Tahap ini bertujuan untuk memberikan gambaran tentang tahapan-tahapan apa saja yang akan dilakukan pada proses penelitian. Teknik yang digunakan pada tahapan pengumpulan data adalah studi literatur atau biasa disebut dengan studi pustaka. Studi literatur atau studi pustaka merupakan sebuah proses pengumpulan data dan informasi berupa teori-teori yang berhubungan dengan masalah yang diteliti. Studi literatur dilakukan dengan mencari jurnal ilmiah, paper ilmiah, dan beberapa tesis yang tersebar di internet. Adapun data-data yang dibutuhkan untuk dalam penelitian ini, diantaranya teori tentang CNN, teori dan proses tentang *Beat Tracking*, teori tentang *Chromagram* dan cara mengubah *audio* ke bentuk *chromagram*, teori tentang bahasa pemrograman Python versi 3, dan beberapa implementasi tentang *backend* pada sebuah aplikasi.

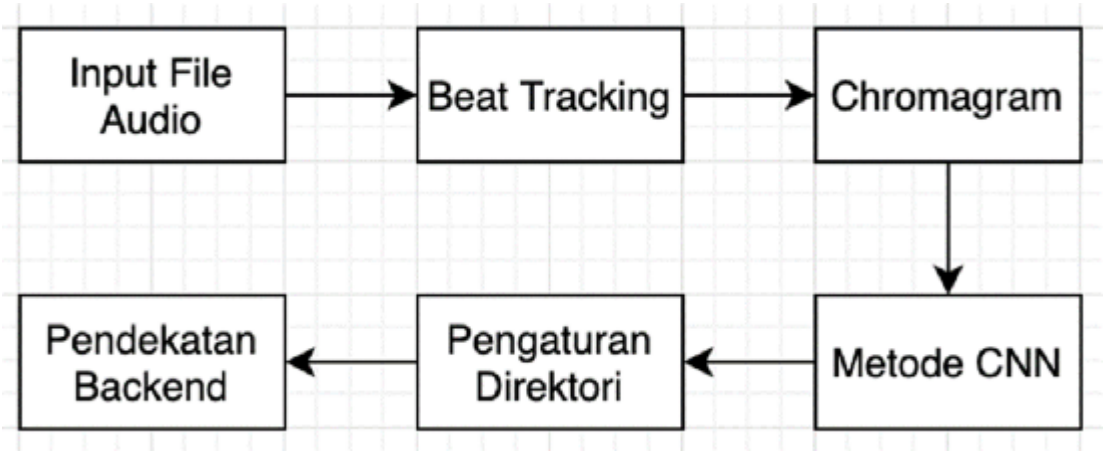
3.1.1 Teori Penelitian

Menurut Kamus Besar Bahasa Indonesia (KBBI), kecerdasan adalah kesempurnaan perkembangan akal budi (seperti kepandaian atau ketajaman pemikiran). Helfi Nasution berpendapat bahwa Kecerdasan Buatan atau dalam bahasa Inggris disebut *Artificial Intelligence* (AI) didefinisikan sebagai kecerdasan yang ditunjukkan oleh suatu entitas buatan (Helfi Nasution, 2012), sedangkan menurut Dedi Nugraha dan Sri Winati (2014), kecerdasan buatan adalah salah satu cabang ilmu pengetahuan yang berhubungan dengan

pemanfaatan mesin untuk memecahkan persoalan yang rumit dengan cara yang lebih manusiawi.

3.2 Kerangka Pemikiran

Kerangka pemikiran dalam penelitian ini memiliki hasil akhir yakni mengetahui akurasi dari metode CNN dalam mengenali sebuah akor dengan memanfaatkan *Chromagram*. Kerangka pemikiran yang akan dilakukan dapat dilihat pada Gambar 3.1.



Gambar 3.1: Kerangka Pemikiran

Langkah awal yang akan dilakukan adalah menerima *input file audio* yang memiliki format .wav. Selanjutnya akan dilakukan *beat tracking*. Pada proses *beat tracking*, *file audio* yang telah diunggah akan dipotong tiap detik dan akan dianalisa bentuk gelombangnya sehingga terbentuklah sebuah onset. Setelah mendapatkan onset dari proses *beat tracking*, selanjutnya onset tersebut akan diubah ke bentuk *chromagram*. Pada proses *chromagram*, onset yang telah didapatkan akan dibentuk ke bentuk gambar. Gambar tersebut nantinya akan dilatih dengan menggunakan metode CNN untuk menemukan akor yang sesuai. Setelah menemukan akor yang sesuai, selanjutnya melakukan proses pengaturan direktori. Proses ini dibutuhkan agar penempatan *parenthesis* dan *child package* bagian *backend* dan *frontend* dapat berkomunikasi satu sama lain. Dan terakhir adalah ada proses pendekatan *backend*. Pendekatan *backend* dibutuhkan agar memudahkan dalam mengakses fungsi yang ada pada pada script Python.

3.2.1 *Audio*

Audio adalah sebuah deret waktu, dimana sumbu y adalah amplitudo arus yang sesuai dengan sebuah membran loudspeaker dan sumbu x adalah sumbu yang sesuai dengan satuan waktu data tersebut. Telinga manusia hanya dapat mendengar bunyi dengan rentang frekuensi antara 20 Hz hingga 20 KHz (20.000Hz). Angka 20 Hz sebagai frekuensi suara terendah yang dapat didengar, sedangkan 20 KHz merupakan frekuensi tertinggi yang dapat didengar. Gelombang suara mengandung sejumlah komponen penting, seperti amplitudo, panjang gelombang, dan frekuensi. Komponen-komponen tersebut mampu membuat suara yang satu berbeda dengan suara yang lain.

Audio yang digunakan dalam penelitian ini adalah audio yang berisi suara piano dan *bass* serta *audio* yang berbentuk format *wav. *Audio* tersebut nantinya akan diolah melalui beberapa tahapan, seperti *Preprocessing* dan *Modeling*. Proses *preprocessing* terbagi atas 2 tahap, yakni *Beat Tracking* dan membentuk gambar *chroma*. Sedangkan untuk proses *modeling* akan langsung dengan menggunakan metode CNN.

3.2.2 *Beat Tracking*

Beat tracking digunakan untuk menentukan contoh waktu dalam rekaman *audio*, di mana pendengar manusia cenderung mengetuk kakinya ke musik. *Beat tracking* pada rancangan menggunakan metode *dynamic programming*. *Dynamic programming* adalah kombinasi pemikiran terstruktur dan pola pikir analitis untuk menyelesaikan sebuah permasalahan. Proses *beat tracking* ini dibutuhkan untuk membentuk ketukan dalam setiap detik dari *file audio* yang diunggah.

3.2.3 *Chromagram*

Chromagram adalah transformasi properti frekuensi-waktu sinyal menjadi *prekursor pitch* yang berubah-ubah untuk sementara waktu. Transformasi ini didasarkan pada pengamatan persepsi tentang sistem pendengaran dan telah terbukti memiliki beberapa sifat matematika yang menarik. *Chromagram* memperluas konsep *chroma* untuk memasukkan dimensi waktu.

Seperti halnya kita menggunakan *spektrogram* untuk menyimpulkan properti tentang distribusi energi sinyal dari frekuensi dan waktu, *chromagram* dapat digunakan untuk menyimpulkan properti tentang distribusi energi sinyal terhadap kroma dan waktu.

3.2.4 *Convolutional Neural Network* (CNN)

Convolutional Neural Network (CNN) adalah salah satu jenis *neural network* yang biasa digunakan pada data *image*. CNN bisa digunakan untuk mendeteksi dan mengenali objek pada sebuah *image*. Metode CNN ini digunakan untuk melatih sebuah sistem dalam mengenali akor pada setiap bar.

3.2.5 Pengaturan Direktori

Tujuan dari pengaturan direktori adalah agar penempatan *parenthesis* dan *child package* bagian *backend* dan *frontend* dapat berkomunikasi satu sama lain. Semua *direktori package* dibuat di dalam direktori berjenis Git. Direktori aplikasi ini dapat diunduh melalui link yang terdapat halaman website Git.

Direktori tersebut akan memiliki tiga buah *direktori child* (sub direktori) dengan nama “app”, “app_settings”, dan “backend”. Selain ketiga sub direktori tersebut, terdapat enam buah file yaitu, “.env”, “.flaskenv”, “.gitignore”, “README.md”, “requirements.txt”, dan “run.py”.

Direktori “app” merupakan bagian dari pendekatan *frontend* yang bertujuan untuk menyediakan tampilan kepada pengguna dengan basis web. Sementara itu, direktori “*backend*” merupakan pendekatan *backend* yang bertujuan untuk menyediakan *model* dan *preprocessing* data masukan. Direktori “app_settings” bertujuan untuk melakukan inisialisasi pengaturan Redis Server dan *session id* dengan tujuan agar kedua pendekatan tersebut dapat berkomunikasi satu sama lain.

File “README.md” dibuat dengan tujuan memberikan penjelasan aplikasi secara singkat pada situs Github. File “requirements.txt” berfungsi untuk menuliskan pustaka Python yang dibutuhkan aplikasi ini. File “requirements.txt” dapat digunakan pengguna untuk melakukan pemasangan pustaka yang dibutuhkan aplikasi dengan memasukan perintah “pip install -r requirements.txt” setelah mengunduh direktori aplikasi dari Github. File terakhir yang ada pada direktori “web” adalah “run.py”. File ini berfungsi untuk menjalankan keseluruhan aplikasi melalui perintah “flask run”.

3.2.6 Pendekatan *Backend*

Pendekatan *backend* pada aplikasi ini ditempatkan pada direktori "*backend*" dan dilengkapi dengan file "__init__.py" di dalamnya. Hal ini dilakukan agar direktori "app" milik pendekatan *frontend* dapat mengakses fungsi yang ada pada pendekatan *backend* dengan melakukan perintah *import backend* pada script Python. Model CNN pada aplikasi menggunakan 1655 citra *chromagram* sebagai data latih, dan 347 citra *chromagram* sebagai data validasi.

Pembuatan model menggunakan pustaka Jupyter, Librosa, dan Keras. Data latih berbentuk wav dan mp3 yang dipisah dan dikelompokan setiap dua ketukan menggunakan pustaka Librosa. Setiap ketukan yang terbentuk berupa detik dengan tipe data *float*. Setiap detik ketukan kemudian disimpan ke dalam file berbentuk ".txt". Setelah membentuk file ".txt", dilakukan *labelling* akor pada masing-masing pasangan ketukan yang dibantu oleh Anastashia Aurelia, pelajar piano dari *Associated Board of the Royal School of Music* (ABRSM) yang sudah berada pada Grade 7.

Akor pada aplikasi berjumlah 24 yang terdiri dari 12 akor major dan 12 akor minor. Setelah *labelling* dilakukan, dilakukan visualisasi citra *spectogram* MFCC berdasarkan detik ketukan, dan akor yang ada pada file ".txt". Visualisasi tersebut kemudian diletakkan di dalam direktori masing-masing kelas untuk dijadikan data latih. Sebanyak 20 persen dari jumlah masing-masing kelas dijadikan data validasi. Data latih dimasukan ke dalam direktori "data_train" dan data validasi ke dalam direktori "data_test". Banyak citra data latih dan data validasi masing-masing kelas dapat dilihat pada Tabel 3.1.

Tabel 3.1: Citra Data Latih Dan Data Validasi

Kelas	Data Latih	Data Validasi
A	48	12
A#	50	9
A#m	248	15
Am	84	27
B	61	14
Bm	53	13
C	76	16
C#	44	11
C#m	77	16
D	79	25
D#	57	13
D#m	24	8
Dm	48	12
Em	56	13
F	56	13
F#	67	13
F#m	54	13
Fm	74	17
G	131	17
G#	54	19
G#m	54	13
Gm	75	16
Jumlah	1655	347

Setelah data latih dan data validasi ditempatkan pada direktori dan kelasnya masing-masing, tahap selanjutnya membuat model CNN menggunakan pustaka Keras. Proses pelatihan menggunakan Jupyter Notebook. Model dilakukan proses kompilasi menggunakan *optimizer* RMSprop yang ada pada pustaka Keras dengan nilai *learning rate* sebesar 0.00001. Tahap selanjutnya dalam pembuatan model adalah dengan memasukan *path* direktori data latih dan data validasi ke dalam Keras. Dimensi citra yang digunakan sebagai data masukan ke dalam model memiliki dimensi 200x200 dan berbentuk tiga dimensi (RGB).

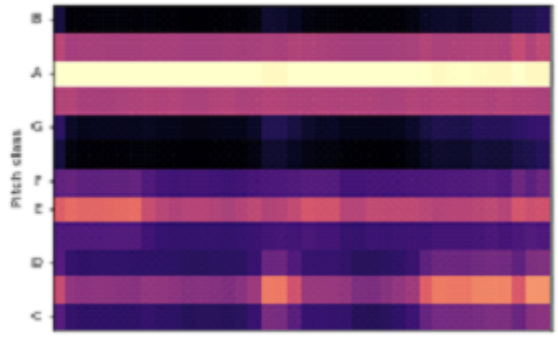
Ukuran *batch* data latih dan data validasi masing-masing adalah sebesar 32 data dan 257 data dengan mode *shuffle* bernilai true, sehingga data latih diproses sebanyak 32 data yang dipilih secara acak oleh Keras dengan jumlah pengulangan sebanyak banyak data latih dibagi besar *batch* yang disebut sebagai *steps per epoch*. Sementara itu, *steps per epoch* yang dipilih untuk data validasi adalah sebanyak dua kali dengan mode *shuffle* bernilai true.

Langkah selanjutnya dalam proses pembuatan model adalah dengan melakukan proses pelatihan. Model yang dilatih disimpan di dalam file berbentuk ".h5" agar dapat digunakan kembali. Proses pelatihan berhasil mendapatkan nilai *accuracy* 87 persen, *loss accuracy* 31 persen, *validation loss* 1.7 persen, dan *validation accuracy* sebanyak 96 persen.

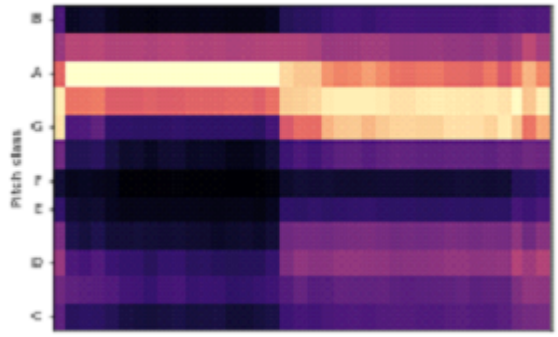
Selanjutnya pengaturan session id. Session id dibentuk pada saat pengguna pertama kali mengunjungi halaman web aplikasi dan juga setelah pengguna menggunakan fitur pengenalan akor. Session id yang dibentuk berguna pada penamaan data masukan pengguna. Penamaan data masukan yang unik berdasarkan session id diperlukan agar data pada saat aplikasi diakses oleh pengguna lebih dari satu tidak saling tumpang tindih.

Redis Server diinisialisasi oleh *backend* terlebih dahulu pada saat aplikasi dijalankan pertama kali. Redis Server pada aplikasi berguna untuk menyimpan session id yang sedang aktif agar dapat diakses baik dari segi *backend* maupun *frontend*. Selain untuk menyimpan session id, Redis Server juga berfungsi sebagai media komunikasi antara *frontend* dengan *backend* pada saat pengguna memilih fitur. Fitur-fitur yang dapat dipilih oleh pengguna adalah memilih data masukan dari direktori pengguna atau melakukan rekaman secara langsung. Perbedaan fitur ini diperlukan suatu variabel pembeda agar fungsi yang dijalankan oleh *backend* tepat sesuai pilihan pengguna.

Model yang sudah dilatih dapat diakses melalui file ".h5" yang sudah disimpan terlebih dahulu pada saat proses pelatihan model, sehingga pada saat pengguna melakukan fitur pengenalan dalam aplikasi, model dapat digunakan kembali. Data masukan pengguna pada saat menggunakan fitur pengenalan pertama-tama dibentuk ".txt" berdasarkan pasangan detik setiap ketukan pada data. Setiap pasangan detik tersebut dibentuk citra *spectogram*. Masing-masing dari citra yang dibentuk kemudian dilakukan proses pengenalan oleh model. Contoh data latih dan data validasi dapat dilihat pada Gambar 3.2 dan Gambar 3.3. Untuk mengetahui akor pada gambar *chromagram*, yang perlu dilihat adalah baris yang paling terang (berwarna putih kekuning-kuningan). Pada data latih, dapat dilihat bahwa akor A yang paling terang dan diikuti oleh akor E dan akor Dm. Setelah dilakukan validasi, ternyata benar bahwa akor yang dilatih adalah akor A. Hal tersebut dapat diketahui dari baris yang paling terang pada data validasi yakni akor A.



Gambar 3.2: Contoh Data Latih Akor A



Gambar 3.3: Contoh Data Validasi Akor A

Bab 4

HASIL DAN PEMBAHASAN

Hasil dari kerangka penelitian adalah mengukur akurasi data akor yang dihasilkan pada *Chord Recognition* dengan menggunakan metode CNN, kemudian menarik kesimpulan dari hasil yang telah didapatkan. Pada bab ini disajikan beberapa hal yang berkaitan dengan proses, hasil, dan pembahasan hasil penelitian, diantaranya, hasil pengujian prediksi model menggunakan *Confusion Matrix*, dan hasil pengujian akurasi akor beserta pembahasannya.

4.1 Pengujian Prediksi Model Menggunakan *Confusion Matrix*

Confusion matrix berbentuk tabel yang sering digunakan untuk memberikan keterangan performa dari model klasifikasi terhadap sekumpulan data tes. Pada aplikasi sistem terdapat 1192 data latih dan 247 data validasi dengan kelas sebanyak 24. Terdapat tiga variabel untuk melakukan perhitungan tingkat akurasi terhadap klasifikasi yang dilakukan oleh model. Ketiga variabel itu adalah *precision*, *recall*, dan *F1-score*. Untuk mendapatkan nilai dari ketiga variabel tersebut dibutuhkan *confusion matrix* yang memiliki nilai-nilai sebagai berikut:

1. *True Negative* (TN)

   TN merupakan jumlah prediksi benar untuk data salah.

2. *True Positive* (TP)

   TP merupakan jumlah prediksi benar untuk data benar.

3. *False Negative* (FN)

   FN merupakan jumlah prediksi salah untuk data salah.

4. *False Positive* (FP)

   FP merupakan jumlah prediksi salah untuk data benar.

4.2 Hasil dan Pembahasan

4.2.1 Hasil Pengujian Model.

Hasil *confusion matrix* pada pengujian dapat dilihat pada Tabel 4.1. Berdasarkan nilai dari *confusion matrix* tersebut, maka nilai *precision*, *recall*, dan *F1-score* dapat dilihat pada Tabel 4.2. Hasil dari pengujian, nilai akurasi yang didapatkan sebesar 89% terhadap 72 data yang diuji. Pada Tabel 4.2 dapat dilihat bahwa prediksi perbedaan akor dapat mudah dilakukan, sementara itu prediksi jenis akor lebih sukar dibedakan. Sebagai contoh untuk kelas akor A#m terdapat kesalahan prediksi menjadi kelas akor A#. Terdapat juga kesalahan prediksi pada kelas akor G#. Model memprediksi kelas akor G# menjadi kelas akor G dan kelas akor G#m. Hal ini dikarenakan akor major dan minor hanya memiliki perbedaan nada kedua sebesar setengah langkah. Akor A# (A# major) terdiri dari nada A#, D, dan F, sedangkan akor A#m (A# minor) terdiri dari nada A#, C#, dan F. Perbedaan nada yang tidak terlalu signifikan mengakibatkan sistem sukar membedakan akor minor dan major.

Tabel 4.1: Hasil *Confusion Matrix*

	A	A#	A#m	Am	B	Bm	C	C#	C#m	Cm	D	D#m	Dm	E	Em	F	F#	F#m	Fm	G	G#	G#m	Gm
A	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
A#	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
A#m	0	1	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Am	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
B	0	0	0	0	6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Bm	0	0	0	0	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
C	0	0	0	0	0	0	3	0	0	1	0	0	0	1	0	0	0	0	0	0	2	0	0
C#	0	0	0	0	0	0	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
C#m	0	0	0	0	0	0	0	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Cm	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0
D	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0	0	0	0	0	0	0	0
D#m	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0	0	0	0	0	0	0
Dm	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0	0	0	0	0	0
E	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0
Em	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0	0	0	0
F	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	3	0	0	1	0	0	0	0
F#	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0	0
F#m	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0
Fm	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0
G	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0
G#	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	2	0
G#m	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	3	0
Gm	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3

Tabel 4.2: Perhitungan *Precision, Recall, F1-score*, dan *Support*

	precision	recall	f1-score	support
A	1.00	1.00	1.00	3
A#	0.67	1.00	0.80	2
A#m	1.00	0.75	0.86	4
Am	0.67	1.00	0.80	2
B	1.00	1.00	1.00	6
Bm	1.00	1.00	1.00	3
C	1.00	0.43	0.60	7
C#	1.00	1.00	1.00	3
C#m	1.00	1.00	1.00	3
Cm	0.67	1.00	0.80	2
D	1.00	1.00	1.00	3
D#m	1.00	1.00	1.00	3
Dm	1.00	1.00	1.00	3
E	0.67	1.00	0.80	2
Em	1.00	1.00	1.00	3
F	1.00	0.60	0.75	5
F#	1.00	1.00	1.00	3
F#m	1.00	1.00	1.00	3
Fm	0.67	1.00	0.80	2
G	1.00	1.00	1.00	3
G#	0.00	0.00	0.00	0
G#m	1.00	0.75	0.86	4
Gm	1.00	1.00	1.00	3
accuracy			0.89	72

4.2.2 Hasil Pengujian Akurasi Akor C Major dan C Minor

Terdapat 30 visualisasi yang didapat dari data lagu masukan akor C dengan nilai F1 yang didapat sebesar 95 persen. Terdapat beberapa kelas akor lain yang diprediksi oleh model yaitu, kelas akor C minor dan kelas akor F major. Kesalahan ini dikarenakan akor C major dan akor C minor hanya memiliki satu buah nada yang berbeda. Akor C major terdiri dari nada C, E, dan G. Sementara akor C minor terdiri dari nada C, D#, dan G. Frekuensi nada E adalah 329.63 Hz sedangkan frekuensi nada D# adalah 311.13 Hz. Kedua frekuensi tersebut saling berdekatan sehingga model salah mengenali akor C major sebagai akor C minor sebanyak dua kali. Detil hasil pengujian pada kelas akor C major dapat dilihat pada Tabel 4.3.

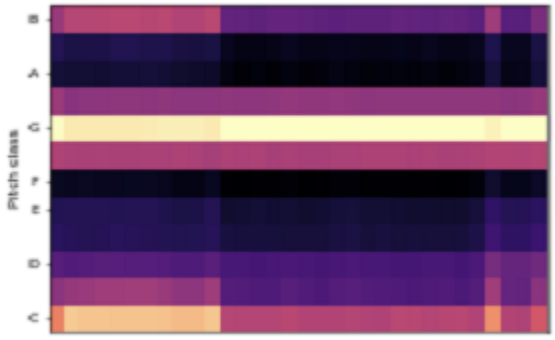
Tabel 4.3: Perhitungan *Precision*, *Recall*, dan *F1-Score* Akor C Major

	Precision	Recall	F1-Score	Support
C	0,90	1,00	0,95	27
Cm	0,00	0,00	0,00	2
F	0,00	0,00	0,00	1
Macro Avg	0,30	0,33	0,32	-
Weighted Avg	0,82	0,90	0,85	-

Pengujian akor C minor mendapatkan nilai F1 sebesar 77 persen dari total 45 data masukan. Model salah memprediksi akor C minor sebanyak 12 prediksi untuk akor C major. Hal ini dikarenakan kemiripan antara komposisi nada yang membentuk akor C minor dan C major saling berdekatan. Selain kesalahan prediksi C major dari model, terdapat juga kesalahan prediksi pada kelas akor D# major. Komposisi nada penyusun akor C minor adalah C, D#, dan G. Komposisi nada penyusun akor D# major adalah D#, G, dan A#. Kesamaan komposisi nada penyusun akor C minor dan D# major adalah nada D# dan G. Perhitungan precision, recall, dan F1-score akor C minor dapat dilihat pada Tabel 4.4. Untuk data latih akor C major dapat dilihat pada Gambar 4.1 dan untuk data validasi C major dapat dilihat pada Gambar 4.2.

Tabel 4.4: Perhitungan *Precision, Recall*, dan *F1-Score* Akor C Minor

	Precision	Recall	F1-Score	Support
Cm	0,62	1,00	0,77	28
C	0,00	0,00	0,00	28
D#	0,00	0,00	0,00	5
Macro Avg	0,21	0,33	0,26	-
Weighted Avg	0,39	0,62	0,48	-



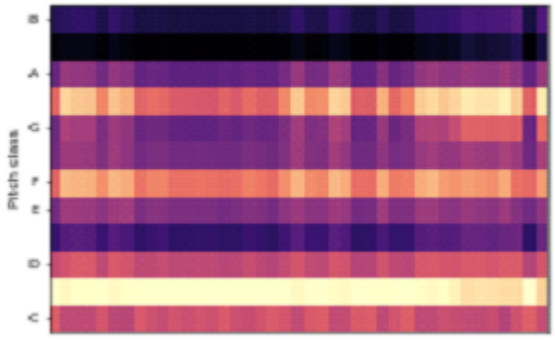
Gambar 4.1: Data Latih Akor C Major

Gambar 4.2: Data Validasi Akor C Major

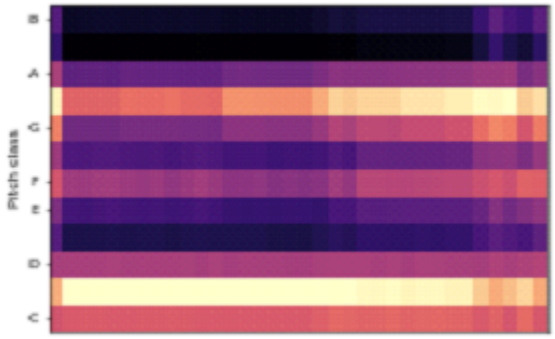
4.2.3 Hasil Pengujian Akurasi Akor C# Major dan C# Minor

Nilai F1 yang diperoleh dari pengujian akor C# major adalah sebesar 91 persen dari total 49 data masukan. Model salah memprediksi akor C# major sebagai akor C# minor dan akor F minor. Komposisi nada penyusun akor C# major adalah C#, F, dan G#. Komposisi nada penyusun akor C# minor adalah C#, E, dan G#. Komposisi nada penyusun akor F minor adalah F, G#, dan C. Kesalahan terbanyak dari prediksi model adalah pada akor F minor dimana terdapat kemiripan komposisi nada penyusun antara kedua akor tersebut. Besar frekuensi nada C adalah 261.63 Hz, sedangkan besar frekuensi nada C# adalah 277.18 Hz. Kedekatan antara kedua nada tersebut memuat model salah memprediksi. Rincian hasil pengujian terhadap akor C# major dapat dilihat pada Tabel 4.5. Untuk data latih akor C# major dapat

dilihat pada Gambar 4.3 dan untuk data validasi C# major dapat dilihat pada Gambar 4.4.



Gambar 4.3: Data Latih C# Major



Gambar 4.4: Data Validasi C# Major

Tabel 4.5: Perhitungan *Precision, Recall*, dan *F1-Score* Akor C# Major

	Precision	Recall	F1-Score	Support
C#	0,84	1,00	0,91	41
C#m	0,00	0,00	0,00	3
Fm	0,00	0,00	0,00	5
Macro Avg	0,28	0,33	0,30	-
Weighted Avg	0,70	0,84	0,76	-

Baris yang terang pada data latih C# major adalah baris akor C# major, akor F minor, dan akor C#m. Setelah dilakukan validasi, hasil baris yang terang adalah baris akor C# major, A minor, dan F. Nilai F1 yang diperoleh dari pengujian kelas akor C# minor adalah sebesar 80 persen. Model salah

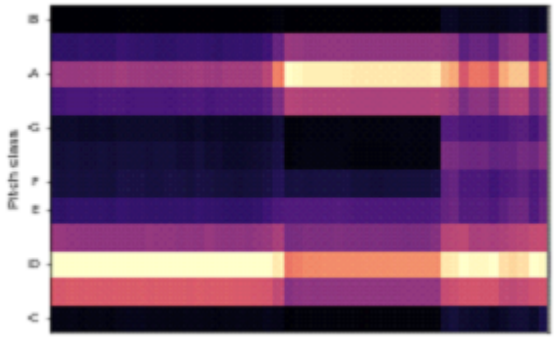
memprediksi akor C# minor sebagai akor C# major dan E major. Kesalahan terbanyak prediksi adalah akor E major yaitu sebesar 12 kesalahan prediksi. Komposisi nada penyusun akor C# minor adalah C#, E, dan G#. Komposisi nada penyusun akor E major adalah E, G#, dan B. Terdapat kesamaan sebanyak dua nada yang ada pada akor C# minor dan E major, yaitu nada E dan G#. Kesalahan pengenalan akor C# minor lainnya adalah kesalahan prediksi kelas akor C# major Kesamaan penyusun nada tersebut membuat model salah memprediksi. Rincian hasil pengujian terhadap akor C# minor dapat dilihat pada Tabel 4.6.

Tabel 4.6: Perhitungan *Precision, Recall*, dan *F1-Score* Akor C# Minor

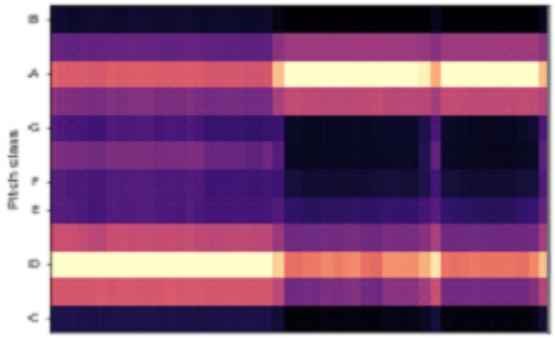
	Precision	Recall	F1-Score	Support
C#m	0,67	1,00	0,80	41
C#	0,00	0,00	0,00	8
Fm	0,00	0,00	0,00	12
Macro Avg	0,22	0,33	0,27	-
Weighted Avg	0,45	0,67	0,54	-

4.2.4 Hasil Pengujian Akurasi Akor D Major dan D Minor

Nilai F1 yang diperoleh dari pengujian kelas akor D major adalah sebesar 83 persen dari total 37 data. Pengujian pada akor D major berhasil memperoleh nilai F1-score sebesar 83 persen. Model salah memprediksi akor D major sebagai akor D minor dan akoe B major. Kesalahan pengenalan akor major dan minor terjadi kembali pada pengujian ini, sedangkan kesalahan mengenali akor D major sebagai B major dikarenakan adanya kedekatan antara kedua komposisi nada penyusun akor-akor tersebut. Komposisi nada penyusun akor D major adalah D, F#, dan A. Komposisi nada penyusun akor B major adalah B, D#, dan F#. Frekuensi nada D dan D# berturut-turut adalah 293.66 Hz dan 311.13 Hz. Rincian hasil pengujian terhadap akor D major dapat dilihat pada Tabel 4.7. Untuk data latih akor D major dapat dilihat pada Gambar 4.5 dan untuk data validasi D major dapat dilihat pada Gambar 4.6.



Gambar 4.5: Data Latih D Major



Gambar 4.6: Data Validasi D Major

Tabel 4.7: Perhitungan *Precision, Recall*, dan *F1-Score* Akor D Major

	Precision	Recall	F1-Score	Support
D	0,70	1,00	0,83	26
B	0,00	0,00	0,00	3
Dm	0,00	0,00	0,00	8
Macro Avg	0,23	0,33	0,28	-
Weighted Avg	0,49	0,70	0,58	-

Nilai F1 yang diperoleh dari hasil pengujian pengenalan kelas akor D minor yaitu sebesar 75 persen dari total 48 data. Model salah memprediksi akor D minor sebagai kelas akor A# major, D major, dan F major. Komposisi nada penyusun akor D minor adalah D, F, dan A. Komposisi nada penyusun A# major adalah A#, D, dan F. Komposisi nada penyusun F major adalah F, A, dan C. Kesalahan pengenalan untuk kelas-kelas tersebut dikarenakan

komposisi nada penyusun antara masing-masing akor sama dan berdekatan. Rincian hasil pengujian terhadap akor D minor dapat dilihat pada Tabel 4.8

Tabel 4.8: Perhitungan *Precision, Recall*, dan *F1-Score* Akor D Minor

	Precision	Recall	F1-Score	Support
D	0,70	1,00	0,83	26
B	0,00	0,00	0,00	3
Dm	0,00	0,00	0,00	8
Macro Avg	0,23	0,33	0,28	-
Weighted Avg	0,49	0,70	0,58	-

4.2.5 Hasil Pengujian Akurasi Akor D# Major dan D# Minor

Nilai F1 yang diperoleh dari hasil pengujian kelas akor D# major yaitu sebesar 81 persen dari total 22 data observasi akor. Model salah mengenali kelas akor D# major sebagai kelas akor B major, D# minor, dan G minor. Kesalahan pengenalan paling banyak adalah mengenali kelas akor G minor sebagai akor D# major. Komposisi nada penyusun akor D# major adalah D#, G, dan A#. Komposisi nada penyusun akor G minor adalah G, A#, dan D. Kemiripan komposisi kedua akor tersebut membuat model salah memprediksi akor D# major sebagai akor G minor sebanyak empat pengenalan. Rincian hasil pengujian terhadap akor D# major dapat dilihat pada Tabel 4.9. Untuk data latih akor D# major dapat dilihat pada Gambar 4.7 dan untuk data validasi D# major dapat dilihat pada Gambar 4.8.

Gambar 4.7: Data Latih D# Major

Gambar 4.8: Data Validasi D# Major

Tabel 4.9: Perhitungan *Precision, Recall*, dan *F1-Score* Akor D# Major

	Precision	Recall	F1-Score	Support
D#	0,68	1,00	0,81	15
B	0,00	0,00	0,00	2
D#m	0,00	0,00	0,00	1
Gm	0,00	0,00	0,00	4
Macro Avg	0,17	0,25	0,20	-
Weighted Avg	0,46	0,68	0,55	-

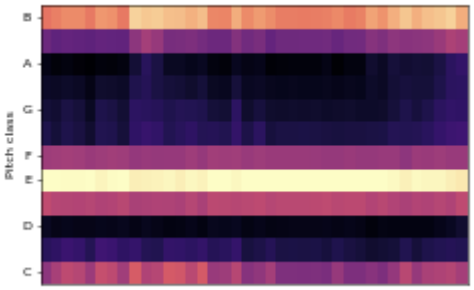
Nilai F1 yang diperoleh dari hasil pengujian kelas akor D# minor adalah sebesar 82 persen dari total 62 data observasi. Model salah memprediksi akor D# minor sebagai akor F# major dan G# major. Kesalahan terbanyak adalah mengenali akor F# major. Pada pengujian akor ini, tidak terdapat kesalahan pengenalan akor minor sebagai akor major dalam root yang sama. Komposisi nada penyusun akor D# minor adalah D#, F#, dan A#. Komposisi nada penyusun akor F# major adalah F#, A#, dan C#. Komposisi nada penyusun akor G# major adalah G#, C, dan D#. Ketiga kelas akor tersebut memiliki nada penyusun akor masing-masing yang mirip sehingga terdapat kesalahan pengenalan dari model pada sistem aplikasi. Rincian hasil pengujian terhadap akor D# minor dapat dilihat pada Tabel 4.10

Tabel 4.10: Perhitungan *Precision, Recall*, dan *F1-Score* Akor D# Minor

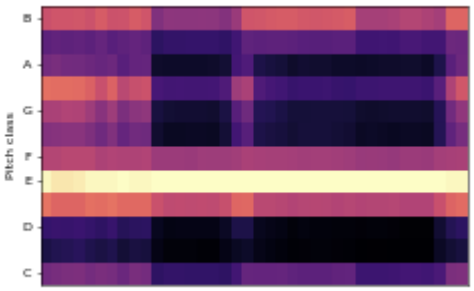
	Precision	Recall	F1-Score	Support
D#m	0,69	1,00	0,82	43
F#	0,00	0,00	0,00	12
G#	0,00	0,00	0,00	7
Macro Avg	0,23	0,33	0,27	-
Weighted Avg	0,48	0,69	0,57	-

4.2.6 Hasil Pengujian Akurasi Akor E Major dan E Minor

Nilai F1 yang diperoleh dari hasil pengujian kelas akor E major adalah sebesar 89 persen dari total 21 data observasi akor. Kesalahan pada pengujian ini dikarenakan kemiripan antara akor major dan akor minor pada root yang sama. Komposisi nada penyusun akor E major adalah E, G#, dan G. Sedangkan, komposisi nada penyusun akor E minor adalah E, G, dan B. Komposisi nada penyusun yang sama antara akor E major dan E minor adalah pada naga E dan B. Selain memiliki kesamaan pada nada E dan B, nada G# dan G memiliki frekuensi yang dekat satu sama lain. Rincian hasil pengujian terhadap akor E major dapat dilihat pada Tabel 4.11. Untuk data latih akor E major dapat dilihat pada Gambar 4.9 dan untuk data validasi E major dapat dilihat pada Gambar 4.10.



Gambar 4.9: Data Latih E Major



Gambar 4.10: Data Validasi E Major

Tabel 4.11: Perhitungan *Precision, Recall*, dan *F1-Score* Akor E Major

	Precision	Recall	F1-Score	Support
E	0,81	1,00	0,89	17
Em	0,00	0,00	0,00	4
Macro Avg	0,40	0,50	0,45	-
Weighted Avg	0,66	0,81	0,72	-

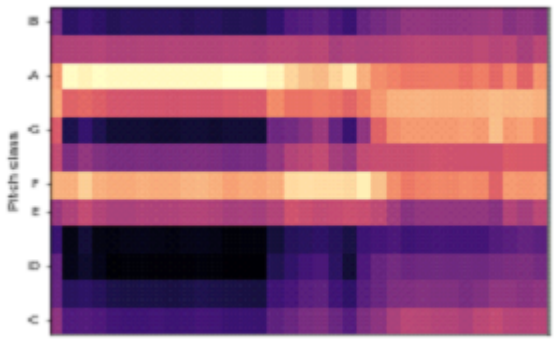
Nilai F1 yang diperoleh dari hasil pengujian kelas akor E minor adalah sebesar 84 persen dari total 18 data masukan observasi. Kesalahan yang didapat dari pengenalan oleh model sistem aplikasi adalah dikarenakan kemiripan antara akor major dan akor minor. Ciri kesalahan yang ada pada pengujian pengenalan kelas akor E minor sama dengan kesalahan pada kelas akor E major dimana ciri komposisi penyusun kedua akor tersebut memiliki keserupaan nada yang tidak jauh. Rincian hasil pengujian terhadap akor E minor dapat dilihat pada Tabel 4.12

Tabel 4.12: Perhitungan *Precision, Recall*, dan *F1-Score* Akor E Minor

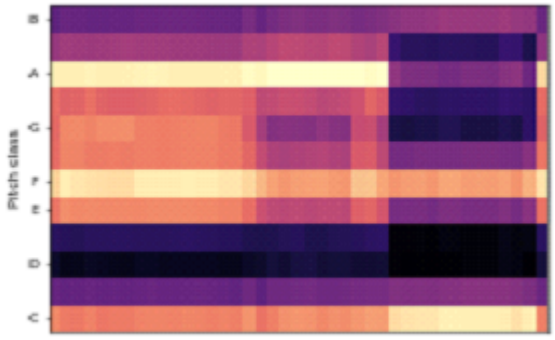
	Precision	Recall	F1-Score	Support
Em	0,72	1,00	0,84	13
E	0,00	0,00	0,00	5
Macro Avg	0,36	0,50	0,42	-
Weighted Avg	0,52	0,72	0,61	-

4.2.7 Hasil Pengujian Akurasi Akor F Major dan F Minor

Nilai F1 pada kelas ini sebesar 99 persen dari total 41 data. Model salah mengenali akor F major sebagai akor A# minor. Pada pengujian ini, model memiliki kesalahan pengenalan yang tidak biasa yaitu A# minor dimana korelasi susunan nada pembentuk akor F major dan A# minor tidak terlalu dekat. Komposisi akor F major terdiri dari nada F, A, dan C. Sedangkan, komposisi akor A# minor terdiri dari nada A#, D, dan F. Kesamaan komposisi nada yang ada pada kedua akor tersebut hanya pada nada F. Selain itu, terdapat juga komposisi kedekatan antara komposisi nada penyusun akor F major dan A# minor yaitu pada nada A dan A#. Kedua nada tersebut berdekatan dengan frekuensi nada A sebesar 440.00 Hz dan frekuensi nada A# sebesar 466.16 Hz. Rincian hasil pengujian terhadap akor F major dapat dilihat pada Tabel 4.13. Untuk data latih akor F major dapat dilihat pada Gambar 4.11 dan untuk data validasi F major dapat dilihat pada Gambar 4.12.



Gambar 4.11: Data Latih F Major



Gambar 4.12: Data Validasi F Major

Tabel 4.13: Perhitungan *Precision, Recall*, dan *F1-Score* Akor F Major

	Precision	Recall	F1-Score	Support
F	0,98	1,00	0,99	40
A#m	0,00	0,00	0,00	1
Macro Avg	0,49	0,50	0,49	-
Weighted Avg	0,95	0,98	0,96	-

Nilai F1 yang diperoleh dari hasil pengujian pengenalan kelas akor F minor adalah sebesar 85 persen dari total 35 data masukan observasi. Kesalahan yang didapatkan dari pengujian ini adalah salah mengenali kelas akor F minor sebagai akor F major. Kesalahan ini diakibatkan oleh kesamaan komposisi susunan akor antara akor major dan akor minor. Model mengenali akor F minor sebagai akor F major yaitu sebanyak 9 data observasi. Rincian hasil pengujian terhadap akor F minor dapat dilihat pada Tabel 4.14.

Tabel 4.14: Perhitungan Precision, Recall, dan F1-Score Akor F Minor

	Precision	Recall	F1-Score	Support
Fm	0,74	1,00	0,85	26
F	0,00	0,00	0,00	9
Macro Avg	0,37	0,50	0,43	-
Weighted Avg	0,55	0,74	0,63	-

4.2.8 Hasil Pengujian Akurasi Akor F# Major dan F# Minor

Nilai F1 pada pengenalan kelas akor F# major sebesar 84 persen dari total 38 data. Kesalahan pengenalan pada pengujian ini adalah A# major dan B major dengan jumlah masing-masing sebesar 6 dan 9 kesalahan. Komposisi nada akor F# major adalah F#, A#, dan C#. Komposisi nada akor A# major adalah A#, D, dan F. Komposisi nada akor B major adalah B, D#, dan F#. Akor F# major dan akor A# major memiliki kemiripan komposisi pada nada A# major dan F dengan F#. Besar frekuensi nada F adalah 249.23 Hz, sedangkan besar frekuensi nada F# adalah 369.99 Hz. Kesalahan pengenalan pada pengujian ini cukup besar pada kelas B major. Kemiripan komposisi nada akor antara akor F# major dengan B major adalah pada nada F# dan A# dengan B. Besar frekuensi A# adalah 466.16 Hz, sedangkan besar frekuensi B adalah 493.88 Hz. Rincian hasil pengujian terhadap akor F# major dapat dilihat pada Tabel 4.15. Untuk data latih akor F# major dapat dilihat pada Gambar 4.13 dan untuk data validasi F# major dapat dilihat pada Gambar 4.14.

Tabel 4.15: Perhitungan *Precision, Recall*, dan *F1-Score* Akor F# Major

	Precision	Recall	F1-Score	Support
F#	0,72	1,00	0,84	38
A#	0,00	0,00	0,00	6
B	0,00	0,00	0,00	9
Macro Avg	0,23	0,33	0,28	-
Weighted Avg	0,51	0,72	0,60	-

Gambar 4.13: Data Latih F# Major

Gambar 4.14: Data Validasi F# Major

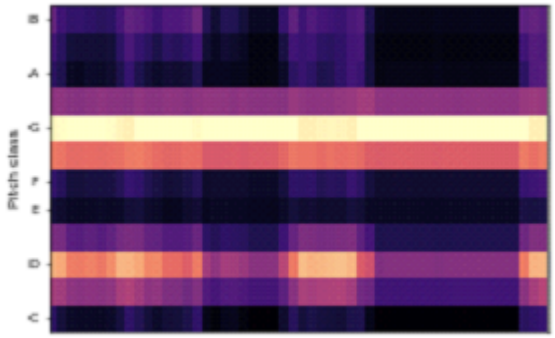
Nilai F1 pada kelas akor ini mendapatkan akurasi sebesar 100 persen dari total 21 data masukan observasi. Model berhasil melakukan pengenalan akor F# minor tanpa terdapat kesalahan seperti yang ada pada kelas lainnya. Rincian hasil pengujian terhadap akor F# minor dapat dilihat pada Tabel 4.16.

Tabel 4.16: Perhitungan Precision, Recall, dan F1-Score Akor F# Minor

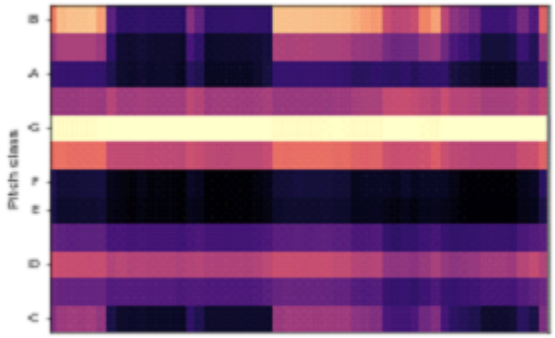
	Precision	Recall	F1-Score	Support
F#m	1,00	1,00	1,00	21
Macro Avg	1,00	1,00	1,00	-
Weighted Avg	1,00	1,00	1,00	-

4.2.9 Hasil Pengujian Akurasi Akor G Major dan G Minor

Nilai F1 yang diperoleh dari pengujian terhadap kelas akor G major adalah sebesar 83 persen dari total 17 data masukan observasi. Model salah mengenali akor G major sebagai akor E minor dan akor G minor. Kemiripan yang ada antara akor major dan akor minor di dalam root yang sama terjadi pada pengujiian akor G major. Rincian hasil pengujian terhadap akor G major dapat dilihat pada Tabel 4.17. Untuk data latih akor G major dapat dilihat pada Gambar 4.15 dan untuk data validasi G major dapat dilihat pada Gambar 4.16.



Gambar 4.15: Data Latih G Major



Gambar 4.16: Data Validasi G Major

Tabel 4.17: Perhitungan *Precision, Recall*, dan *F1-Score* Akor G Major

	Precision	Recall	F1-Score	Support
G	0,71	1,00	0,83	12
Em	0,00	0,00	0,00	1
Gm	0,00	0,00	0,00	4
Macro Avg	0,24	0,33	0,28	-
Weighted Avg	0,50	0,71	0,58	-

Pada pengujian akor G minor, Model berhasil memberikan pengenalan dengan nilai F1 sebesar 98 persen dari total 42 data. Pada pengujian ini, model memiliki kesalahan pengenalan yaitu akor C major. Akor G minor terdiri dari nada G, A#, dan D. Sedangkan, akor C major terdiri dari nada C, E, dan G. Kesamaan antara akor G minor dan akor C major hanya satu nada yaitu nada G. Kesalahan yang ada pada pengujian ini sama dengan

kesalahan pada pengujian F major, dimana kesalahan pengenalan memiliki korelasi antara komposisi kedua akor yang tidak begitu jauh antara nilai akor sesungguhnya dan nilai akor pada pengenalan yang salah. Rincian hasil pengujian terhadap akor G minor dapat dilihat pada Tabel 4.18.

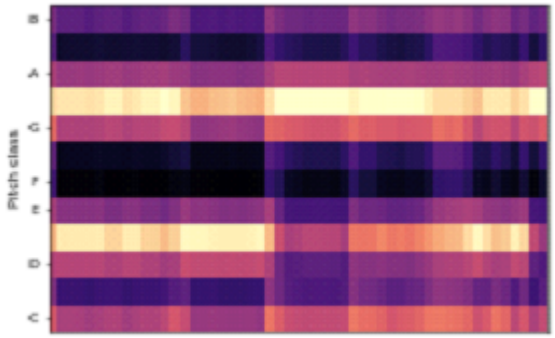
Tabel 4.18: Perhitungan *Precision, Recall,* dan *F1-Score* Akor G Minor

	Precision	Recall	F1-Score	Support
Gm	0,95	1,00	0,98	40
C	0,00	0,00	0,00	2
Macro Avg	0,48	0,50	0,49	-
Weighted Avg	0,91	0,95	0,93	-

4.2.10 Hasil Pengujian Akurasi Akor G# Major dan G# Minor

Pengujian ini berhasil memperoleh nilai F1 sebesar 92 persen dari total 35 data. Kesalahan pengenalan pada pengujian akor G# major adalah C major dan F minor. Komposisi nada pada akor G# major adalah G#, C, dan D#. Komposisi nada akor C major adalah C, E, dan G. Sedangkan, komposisi nada akor F minor adalah F, G#, dan C. Terdapat kemiripan komposisi akor G# major dengan F minor yaitu pada nada G# dan C. Sedangkan, kemiripan komposisi antara akor G# major dan C major terletak pada nada C dan G dengan G#. G memiliki besar frekuensi 392.00 Hz dan G# memiliki besar frekuensi 415.30 Hz. Kedua nada tersebut sangat dekat sehingga model CNN pada sistem aplikasi salah mengenali G# major dengan C major. Rincian hasil pengujian terhadap akor G# major dapat dilihat pada Tabel 4.19. Untuk data latih akor G# major dapat dilihat pada Gambar 4.17 dan untuk data validasi G# major dapat dilihat pada Gambar 4.18.

Gambar 4.17: Data Latih G# Major



Gambar 4.18: Data Validasi G# Major

Tabel 4.19: Perhitungan *Precision, Recall*, dan *F1-Score* Akor G# Major

	Precision	Recall	F1-Score	Support
Gm	0,86	1,00	0,92	30
Fm	0,00	0,00	0,00	4
F	0,00	0,00	0,00	1
Macro Avg	0,29	0,33	0,31	-
Weighted Avg	0,73	0,86	0,79	-

Nilai F1 yang diperoleh dari pengujian kelas akor G# minor adalah sebesar 86 persen dari total 25 data masukan observasi. Model salah memprediksi akor G# minor sebagai akor G# major dikarenakan kemiripan antara akor major dan minor dalam root yang sama. Kesalahan pengenalan lainnya adalah mengenali kelas akor G# minor sebagai kelas akor B major. Komposisi nada penyusun akor G# minor adalah G#, B, dan D#. Komposisi nada

penyusun akor B major adalah B, D#, dan F#. Terdapat dua buah nada yang sama antara kedua penyusun akor tersebut. Rincian hasil pengujian terhadap akor G# minor dapat dilihat pada Tabel 4.20.

Tabel 4.20: Perhitungan *Precision, Recall*, dan *F1-Score* Akor G# Minor

	Precision	Recall	F1-Score	Support
G#m	0,76	1,00	0,86	19
B	0,00	0,00	0,00	4
C#m	0,00	0,00	0,00	2
Macro Avg	0,25	0,33	0,29	-
Weighted Avg	0,58	0,76	0,66	-

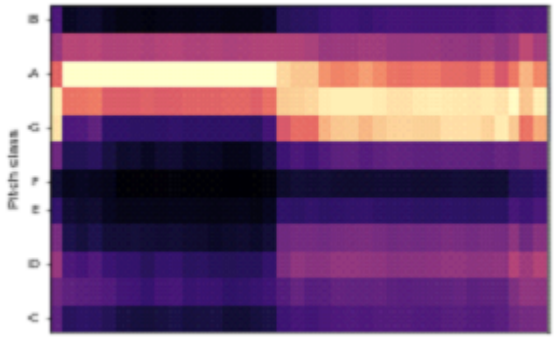
4.2.11 Hasil Pengujian Akurasi Akor A Major dan A Minor

Nilai F1 yang didapat dari pengujian pengenalan akor A major adalah sebesar 98 persen dari total 32 data masukan observasi. Pada pengeujian kelas akor ini, kesalahan pengenalan yang didapat sebanyak satu buah data observasi. Model salah mengenali akor A major sebagi akor A minor dikarenakan kemiripan antara nada penyusun akor major dan akor minor dalam root yang sama. Komposisi nada penyusun akor A major adalah A, C#, dan E. Sedangkan, komposisi nada penyusun akor A minor adalah A, C, dan E. Frekuensi nada C# adalah 277.18 Hz dan frekuensi nada C adalah 261.63 Hz. Kedekatan kedua frekuensi tersebut membuat perbedaan kedua kelas akor yang dekat antara satu sama lain dan menyebabkan model salah memprediksi. Rincian hasil pengujian terhadap akor A major dapat dilihat pada Tabel 4.21. Untuk data latih akor A major dapat dilihat pada Gambar 4.19 dan untuk data validasi A major dapat dilihat pada Gambar 4.20.

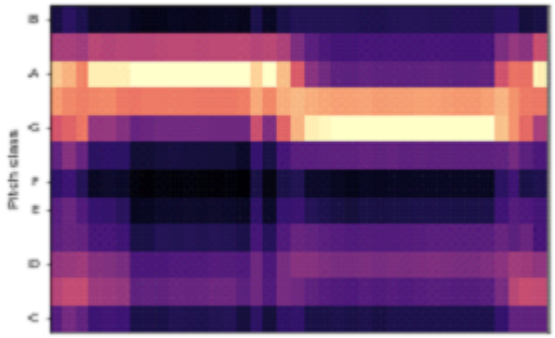
Tabel 4.21

Tabel 4.21: Perhitungan *Precision, Recall*, dan *F1-Score* Akor A Major

	Precision	Recall	F1-Score	Support
A	0,97	1,00	0,98	31
Am	0,00	0,00	0,00	1
Macro Avg	0,48	0,50	0,49	-
Weighted Avg	0,94	0,97	0,95	-



Gambar 4.19: Data Latih A Major



Gambar 4.20: Data Validasi A Major

Nilai F1 yang didapat dari pengujian pengenalan kelas akor A minor adalah sebesar 80 persen dari total 24 data masukan observasi. Model salah mengenali akor A minor sebagai akor A major sebanyak 6 data observasi. Selain kesalahan tersebut, model juga salah mengenali akor A minor sebagai akor C major sebanyak dua data observasi. Kesalahan pengenalan akor A minor pada pengujian ini dikarenakan kemiripan komposisi nada susunan akor-akor tersebut. Rincian hasil pengujian terhadap akor A minor dapat dilihat pada Tabel 4.22.

Tabel 4.22: Perhitungan *Precision, Recall*, dan *F1-Score* Akor A Minor

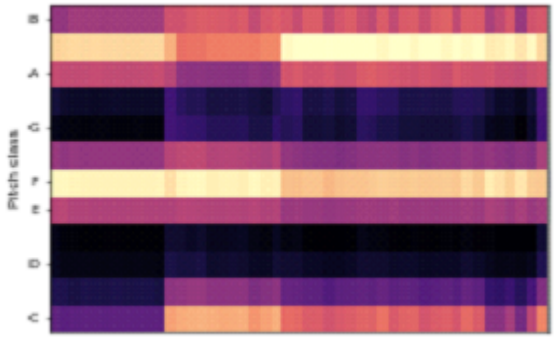
	Precision	Recall	F1-Score	Support
Am	0,67	1,00	0,80	16
A	0,00	0,00	0,00	6
C	0,00	0,00	0,00	2
Macro Avg	0,22	0,33	0,27	-
Weighted Avg	0,44	0,67	0,55	-

4.2.12 Hasil Pengujian Akurasi Akor A# Major dan A# Minor

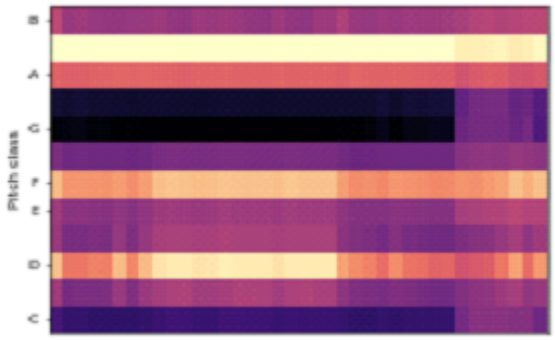
Nilai F1 yang diperoleh dari pengujian pengenalan kelas akor A# major adalah sebesar 82 persen dari total 43 data masukan observasi. Model salah mengenali akor A# major sebagai akor A# minor dan akor D minor. Komposisi nada penyusun akor A# major adalah A#, D, dan F. Komposisi nada penyusun akor A# minor adalah A#, C#, dan F. Komposisi nada penyusun akor D minor adalah D, F, dan A. Kesamaan komposisi nada akorskor tersebut membuat perbedaan ciri antar akor tersebut tidak jauh dan menyebabkan model salah mengenali akor A# major sebanyak 30 persen pada pengujian ini. Rincian hasil pengujian terhadap akor A# major dapat dilihat pada Tabel 4.23. Untuk data latih akor A# major dapat dilihat pada Gambar 4.21 dan untuk data validasi A# major dapat dilihat pada Gambar 4.22.

Tabel 4.23: Perhitungan *Precision, Recall*, dan *F1-Score* Akor A# Major

	Precision	Recall	F1-Score	Support
A#	0,70	1,00	0,82	30
A#m	0,00	0,00	0,00	8
Dm	0,00	0,00	0,00	5
Macro Avg	0,23	0,33	0,27	-
Weighted Avg	0,49	0,70	0,57	-



Gambar 4.21: Data Latih A# Major



Gambar 4.22: Data Validasi A# Major

Tabel 4.24: Perhitungan *Precision, Recall*, dan *F1-Score* Akor A# Minor

	Precision	Recall	F1-Score	Support
A#m	1,00	1,00	1,00	41
Macro Avg	1,00	1,00	1,00	-
Weighted Avg	1,00	1,00	1,00	-

4.2.13 Hasil Pengujian Akurasi Akor B Major dan B Minor

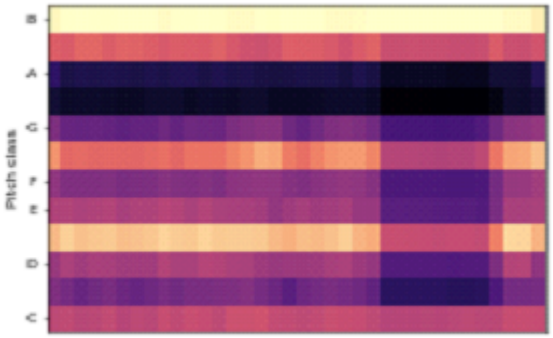
Nilai F1 yang didapat dari hasil pengujian pengenalan kelas akor B major adalah sebesar 86 persen dari total 54 data masukan observasi. Model salah memprediksi kelas akor B major sebagai akor D# minor dan G minor. Komposisi nada penyusun akor B major adalah B, D#, dan F#. Komposisi nada penyusun akor D# minor adalah D#, F#, dan B#. Komposisi nada penyusun akor G minor adalah G, A#, dan D. Kesamaan komposisi nada penyusun

antara akor B major dan akor D# minor adalah pada nada D# dan F#. Sementara itu, tidak ada kesamaan pada komposisi penyusun akor B major dan akor G minor. Namun, kedua akor tersebut mempunyai komposisi nada penyusun akor yang berdekatan. Nada-nada tersebut adalah A#, D, dan G. Ketiga nada yang ada didalam akor G minor semuanya berdekatan dengan ketiga nada yang ada pada akor B major. Kedekatan ini yang membuat model salah melakukan prediksi sebanyak 6 buah data observasi. Rincian hasil pengujian terhadap akor B major dapat dilihat pada Tabel 4.25. Untuk data latih akor B major dapat dilihat pada Gambar 4.23 dan untuk data validasi B major dapat dilihat pada Gambar 4.24.

Tabel 4.25: Perhitungan *Precision, Recall*, dan *F1-Score* Akor B Major

	Precision	Recall	F1-Score	Support
B	0,76	1,00	0,86	41
D#m	0,00	0,00	0,00	7
Gm	0,00	0,00	0,00	6
Macro Avg	0,25	0,33	0,29	-
Weighted Avg	0,58	0,76	0,66	-

Gambar 4.23: Data Latih B Major



Gambar 4.24: Data Validasi B Major

Nilai F1 yang didapat dari hasil pengujian kelas akor B minor adalah sebesar 86 persen dari 45 data observasi pengenalan. Terdapat kesalahan pengenalan akor B minor pada model yaitu, mengenali akor B minor sebagai akor D major dan akor B major. Kesalahan mengenali akor B minor menjadi akor B major dikarenakan kemiripan ciri akor major dan minor pada root yang sama. Komposisi nada penyusun akor B minor adalah B, D, dan F#. Komposisi nada penyusun akor D major adalah D, F#, dan A. Perbedaan komposisi nada penyusun akor B minor dan akor D major hanya pada nada B dan nada A. Kemiripan ciri komposisi inilah yang membuat model salah mengenali akor B minor sebagai akor D major. Rincian hasil pengujian terhadap akor B minor dapat dilihat pada Tabel 4.26.

Tabel 4.26: Perhitungan *Precision, Recall*, dan *F1-Score* Akor B Minor

	Precision	Recall	F1-Score	Support
Bm	0,76	1,00	0,86	34
B	0,00	0,00	0,00	7
D	0,00	0,00	0,00	4
Macro Avg	0,25	0,33	0,29	-
Weighted Avg	0,57	0,76	0,65	-

4.2.14 Hasil Pengujian Pengenalan Akor Data Lagu

Pengujian selanjutnya adalah menggunakan lagu “Fly Me To The Moon”. Akor-akor yang digunakan pada lagu “Fly Me To The Moon” adalah A minor, C major, D minor, E minor, F major, dan G major. Lagu ini dimainkan dengan nada dasar pada C major dengan ketukan 4/4. Hasil ratarata nilai F1 yang didapat dari pengujian pengenalan akor lagu “Fly Me To The Moon” adalah

sebesar 59,33 persen dari total 64 data observasi pengenalan. Nilai F1 pengenalan untuk kelas akor A minor, C major, D minor, E minor, F major, dan G major berturut-turut adalah 67 persen, 42 persen, 77 persen, 55 persen, 48 persen, dan 67 persen. Rincian hasil akurasi pengujian pengenalan akor lagu "Fly Me To The Moon" dapat dilihat pada Tabel 4.27 dan jumlah kesalahan prediksi pengujian pengenalan akor lagu "Fly Me To The Moon" dapat dilihat pada Tabel 4.28.

Tabel 4.27: Perhitungan *Precision, Recall*, dan *F1-Score* Pengujian Lagu "Fly Me To The Moon"

Y True	Precision	Recall	F1-Score	Total
Am	1,00	0,44	0,61	16
C	0,71	0,50	0,59	10
Dm	1,00	0,50	0,67	4
Em	1,00	0,60	0,75	5
F	0,55	0,35	0,43	17
G	1,00	0,62	0,77	29
Macro Avg	0,38	0,22	0,27	-
Weighted Avg	0,87	0,51	0,64	-

Tabel 4.28: Jumlah Kesalahan Prediksi Pengujian Pengenalan Akor Lagu "Fly Me To The Moon"

Y True	Y False	Jumlah Kesalahan
Am	A	7
	C	2
C	Cm	3
	F	2
Dm	D	1
	F	1
Em	E	2
F	Fm	7
	C	4
G	Gm	18

4.2.15 Hasil Rinci Pengujian

Dari pengujian akurasi yang telah dilakukan, dapat disimpulkan bahwa pengujian akurasi pada masing-masing kelas memberikan nilai F1 yang cukup baik. Besar nilai F1 pada masing-masing kelas dapat dilihat pada Tabel 4.29

Tabel 4.29: Rincian Nilai F1 Keseluruhan Pengujian Pengenalan Akor Berdasarkan Root

Nomor	Kelas Akor	Nilai F1
1	C	0,95
2	Cm	0,77
3	C#	0,91
4	C#m	0,80
5	D	0,83
6	Dm	0,75
7	D#	0,81
8	D#m	0,82
9	E	0,89
10	Em	0,84
11	F	0,99
12	Fm	0,85
13	F#	0,84
14	F#m	1,00
15	G	0,83
16	Gm	0,98
17	G#	0,92
18	G#m	0,86
19	A	0,98
20	Am	0,80
21	A#	0,82
22	A#m	1,00
23	B	0,86
24	Bm	0,86
Rata-rata		0,87

Bab 5

KESIMPULAN DAN SARAN

5.1 Kesimpulan

Dari pengujian akurasi yang telah dilakukan, dapat disimpulkan bahwa pengujian akurasi pada masing-masing kelas memberikan nilai F1 yang cukup baik. Nilai F1 yang paling besar diperoleh pada pengujian akor F# minor dan A# minor yaitu sebesar 100 persen. Nilai F1 yang paling kecil diperoleh pada pengujian akor D minor yaitu sebesar 75 persen. Rata-rata nilai F1 dari keseluruhan pengujian pengenalan akor berdasarkan nada root adalah sebesar 87 persen.

5.2 Saran

Saran yang dapat diberikan untuk memperluas penelitian ini adalah sebagai berikut:

1. Meneliti pengenalan akor dari suara alat musik selain *piano* dan *bass*.
2. Melakukan penelitian menggunakan metode *KNN* dan membandingkan hasilnya dengan metode *CNN*.

DAFTAR PUSTAKA

Abhirawa, H., Jondri, and Arifianto, A. (2017). Pengenalan wajah menggunakan convolutional neural network. *Journal E-Proceeding of Engineering*, 4(3):4907 – 4916.

Clarissa, H., Hidayat, B., and Suhardjo. (2018). Pengolahan citra deteksi kista melalui periapical radiograf dengan metode local binary pattern dan learning vector quantization. *Journal E-Proceeding of Engineering*.

Collins, N. (2011). Scmir: A supercollider music information retrieval library. *Brighton : University of Sussex*.

Falah, Adnan Hassal, d. J. (2019). Klasifikasi suara paru normal dan abnormal menggunakan deep neural network dan support vector machine. *Journal E-Proceeding of Engineering*.

Fatihah, N., Karna, N., and Patmasari, R. (2019). Evaluation of dlx microprocessor instructions efficiency for image compression. *Journal E-Proceeding of Engineering*.

Gumilar, T., Suwandi, and Bethaningtyas, H. (2015). Deteksi kesalahan nada pada string gitar dengan menggunakan harmonic product spectrum. *Journal E-Proceeding of Engineering*.

Hakim, D. M. and Rainarli, E. (2019). Convolutional neural network untuk pengenalan citra notasi musik. *Techno.COM*.

Ibrahim, H.S., J. and Wlsesty, U. (2018). Analisis deep learning untuk mengenali qrs kompleks pada sinyal ecg dengan metode cnn. *Journal E-Proceeding of Engineering*.

Krebs, F., Bock, S., and Widmer, G. (2013). Rhythmic pattern modeling for beat and downbeat tracking in musical audio. *Austria : Johannes Kepler University*.

Kumalasari, D., Novamizanti, L., and Ramatryana, I. (2019). Penentuan lokasi chorus pada musik mp3 menggunakan koefiesien korelasi 2-d pada frame berbasis ciri mel-frequency cepstral coeffisient (mfcc). *Journal E-Proceeding of Engineering*.

Laksana, E. and Sulianta, F. (2017). Analisis dan studi komparatif algoritma klasifikasi genre musik. *STMIK AMIKOM Yogyakarta : diterbitkan*.

Lionel, D., Adipranata, R., and Setyati, E. (2019). Klasifikasi genre musik menggunakan metode deep learning convolutional neural network dan mel-spektrogram. *Universitas Kristen Petra : diterbitkan*.

Maharani, M., Hidayat, B., and Suhardjo. (2018). Perbandingan deteksi pulpitis melalui citra radiograf periapikal dengan ekstrasi ciri watershed dan grey level co-occurrence matrix (glcm) dengan klasifikasi k-nearest neighbour (k-nn). *Journal E-Proceeding of Engineering*.

Mahmud, K. and Adiwijaya, Al Faraby, S. (2019). Klasifikasi citra multi-kelas menggunakan convolutional neural network. *Journal E-Proceeding of Engineering*.

Parlys, A., Zahra, A., and Hidayatno, A. (2018). Penggolongan lagu berdasarkan spektogram dengan convolution neural network. *Transient*.

Permana, T. G., Hidayat, B., and Susatio, E. (2014). Identifikasi akor gitar menggunakan algoritma harmonic product spectrum. *Journal E-Proceeding of Engineering*.

Putra, I. W. S. E. (2016). Klasifikasi citra menggunakan convolutional neural network (cnn) pada caltech 101. *Surabaya : Institut Teknologi Sepuluh Nopember*.

Waskito, T., Sony, S., and Setianingsih, C. (2019). Kendali robot beroda dengan gerak isyarat tangan berbasis image processing. *Journal E-Proceeding of Engineering*.

Zulfikar, M. M. M. (2015). Pengenalan dan representasi simbol akor musik menggunakan hidden markov model dengan pendekatan doubly nested circle of fifth. *Telkom University*.

Lampiran 1

Source Code

```
{
 "cells": [
  {
   "cell_type": "code",
   "execution_count": 1,
   "metadata": {},
   "outputs": [
    {
     "name": "stderr",
     "output_type": "stream",
     "text": "Using TensorFlow backend.\n"
    }
   ],
   "source": "from keras.models import Sequential\nfrom keras.
      layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout,
       Activation, BatchNormalization"
  },
  {
   "cell_type": "code",
   "execution_count": 2,
   "metadata": {},
   "outputs": [],
   "source": "import numpy as np\nimport pandas as pd\nimport
      matplotlib.pyplot as plt\nimport tensorflow\n%matplotlib
      inline"
  },
  {
   "cell_type": "code",
   "execution_count": 3,
   "metadata": {},
   "outputs": [
    {
     "name": "stdout",
     "output_type": "stream",
```

```
   "text": "Model: \"sequential_1\"\
      n_________________________________________________________________
      \nLayer (type)                 Output Shape
      Param #   \n
      =================================================================\
      nconv2d_1 (Conv2D)            (None, 198, 198, 32)
      896       \
      n_________________________________________________________________
      \nconv2d_2 (Conv2D)            (None, 196, 196, 16)
      4624      \
      n_________________________________________________________________
      \nmax_pooling2d_1 (MaxPooling2 (None, 65, 65, 16)        0
               \
      n_________________________________________________________________
      \nflatten_1 (Flatten)          (None, 67600)             0
               \
      n_________________________________________________________________
      \ndense_1 (Dense)              (None, 12)
      811212    \
      n_________________________________________________________________
      \ndropout_1 (Dropout)          (None, 12)                0
               \
      n_________________________________________________________________
      \ndense_2 (Dense)              (None, 24)
      312       \n
      =================================================================\
      nTotal params: 817,044\nTrainable params: 817,044\nNon-
      trainable params: 0\
      n_________________________________________________________________
      \n"
   }
  ],
  "source": "nb_classes = 24\nmodel = Sequential()\nmodel.add(
     Conv2D(32, kernel_size=(3,3), activation='relu', input_shape
     =(200,200,3)))\nmodel.add(Conv2D(16, kernel_size=(3,3),
     activation='relu', input_shape=(200,200,3)))\nmodel.add(
     MaxPooling2D(pool_size=(3, 3)))\nmodel.add(Flatten())\nmodel
     .add(Dense(12, activation= 'relu'))\nmodel.add(Dropout(0.2))
     \nmodel.add(Dense(nb_classes, activation = 'softmax'))\n#
     model.layers[0].trainable = True\nmodel.summary()"
 },
 {
  "cell_type": "code",
  "execution_count": 8,
  "metadata": {},
```

```
 "outputs": [],
 "source": "from keras.optimizers import RMSprop\noptimizers =
    RMSprop(lr=0.00001)\nmodel.compile(\n    optimizer=
    optimizers,\n    loss='categorical_crossentropy',\n
    metrics=['accuracy']\n)"
},
{
 "cell_type": "code",
 "execution_count": 9,
 "metadata": {},
 "outputs": [
  {
   "name": "stdout",
   "output_type": "stream",
   "text": "Found 1712 images belonging to 24 classes.\nFound
      318 images belonging to 24 classes.\n"
  }
 ],
 "source": "from keras.preprocessing.image import
    ImageDataGenerator\nfrom keras.models import Sequential\
    nfrom keras.layers import Dropout, Flatten, Dense\nfrom
    keras import applications\ntrain_datagen =
    ImageDataGenerator(rescale=1./255,\n
                                    horizontal_flip=False)\
    ntest_datagen = ImageDataGenerator(rescale=1./255)\nclasses
    = ['A','A#','A#m','Am','B','Bm',\n          'C','C#','C#m
    ','Cm','D','D#',\n          \"D#m\",'Dm','E','Em','F','F
    #',\n          'F#m','Fm','G','G#','G#m','Gm']\
    ntrain_generator = train_datagen.flow_from_directory(\n
          'train/data_train', \n        target_size=(200, 200)
    ,\n        shuffle = True,\n        batch_size=64,       \n
          class_mode='categorical')\ntest_generator =
    test_datagen.flow_from_directory(\n        'train/data_test
    ',\n        target_size=(200, 200),\n        shuffle = True
    ,\n        batch_size=247,\n        class_mode='categorical
    ')"
},
{
 "cell_type": "code",
 "execution_count": 10,
 "metadata": {},
 "outputs": [],
 "source": "from keras.callbacks import ModelCheckpoint\
    nfilepath=\"weight.checkpoint5.cont.h5\"\ncheckpoint = [
    ModelCheckpoint(filepath, monitor='val_loss', verbose=1,
```

```
    save_best_only=True, mode='min'),\n
    ModelCheckpoint(filepath, monitor='loss', verbose=1,
    save_best_only=True, mode='min')]"
},
{
 "cell_type": "code",
 "execution_count": 11,
 "metadata": {
  "scrolled": true
 },
 "outputs": [
  {
   "name": "stdout",
   "output_type": "stream",
   "text": "Epoch 1/1000\n26/26 [==============================]
       - 60s 2s/step - loss: 3.0627 - accuracy: 0.1438 -
      val_loss: 3.1323 - val_accuracy: 0.0409\n\nEpoch 00001:
      val_loss improved from inf to 3.13228, saving model to
      weight.checkpoint5.cont.h5\n\nEpoch 00001: loss improved
      from inf to 3.06291, saving model to weight.checkpoint5.
      cont.h5\nEpoch 2/1000\n26/26
      [==============================] - 57s 2s/step - loss:
      3.0615 - accuracy: 0.1456 - val_loss: 3.2120 -
      val_accuracy: 0.0409\n\nEpoch 00002: val_loss did not
      improve from 3.13228\n\nEpoch 00002: loss improved from
      3.06291 to 3.06162, saving model to weight.checkpoint5.
      cont.h5\nEpoch 3/1000\n26/26
      [==============================] - 56s 2s/step - loss:
      3.0638 - accuracy: 0.1468 - val_loss: 3.2062 -
      val_accuracy: 0.0409\n\nEpoch 00003: val_loss did not
      improve from 3.13228\n\nEpoch 00003: loss did not improve
      from 3.06162\nEpoch 4/1000\n26/26
      [==============================] - 71s 3s/step - loss:
      3.0567 - accuracy: 0.1462 - val_loss: 3.1539 -
      val_accuracy: 0.0409\n\nEpoch 00004: val_loss did not
      improve from 3.13228\n\nEpoch 00004: loss improved from
      3.06162 to 3.05760, saving model to weight.checkpoint5.
      cont.h5\nEpoch 5/1000\n26/26
      [==============================] - 56s 2s/step - loss:
      3.0700 - accuracy: 0.1390 - val_loss: 3.1996 -
      val_accuracy: 0.0409\n\nEpoch 00005: val_loss did not
      improve from 3.13228\n\nEpoch 00005: loss did not improve
      from 3.05760\nEpoch 6/1000\n26/26
      [==============================] - 78s 3s/step - loss:
      3.0613 - accuracy: 0.1448 - val_loss: 3.1987 -
```

val_accuracy: 0.0409\n\nEpoch 00006: val_loss did not improve from 3.13228\n\nEpoch 00006: loss did not improve from 3.05760\nEpoch 7/1000\n26/26
[==============================] - 76s 3s/step - loss: 3.0699 - accuracy: 0.1379 - val_loss: 3.1736 - val_accuracy: 0.0409\n\nEpoch 00007: val_loss did not improve from 3.13228\n\nEpoch 00007: loss did not improve from 3.05760\nEpoch 8/1000\n26/26
[==============================] - 62s 2s/step - loss: 3.0619 - accuracy: 0.1490 - val_loss: 3.1940 - val_accuracy: 0.0409\n\nEpoch 00008: val_loss did not improve from 3.13228\n\nEpoch 00008: loss did not improve from 3.05760\nEpoch 9/1000\n26/26
[==============================] - 56s 2s/step - loss: 3.0475 - accuracy: 0.1566 - val_loss: 3.1843 - val_accuracy: 0.0409\n\nEpoch 00009: val_loss did not improve from 3.13228\n\nEpoch 00009: loss improved from 3.05760 to 3.04834, saving model to weight.checkpoint5.cont.h5\nEpoch 10/1000\n26/26
[==============================] - 56s 2s/step - loss: 3.0682 - accuracy: 0.1403 - val_loss: 3.2236 - val_accuracy: 0.0409\n\nEpoch 00010: val_loss did not improve from 3.13228\n\nEpoch 00010: loss did not improve from 3.04834\nEpoch 11/1000\n26/26
[==============================] - 56s 2s/step - loss: 3.0626 - accuracy: 0.1426 - val_loss: 3.1664 - val_accuracy: 0.0409\n\nEpoch 00011: val_loss did not improve from 3.13228\n\nEpoch 00011: loss did not improve from 3.04834\nEpoch 12/1000\n26/26
[==============================] - 55s 2s/step - loss: 3.0682 - accuracy: 0.1424 - val_loss: 3.1899 - val_accuracy: 0.0409\n\nEpoch 00012: val_loss did not improve from 3.13228\n\nEpoch 00012: loss did not improve from 3.04834\nEpoch 13/1000\n26/26
[==============================] - 61s 2s/step - loss: 3.0583 - accuracy: 0.1499 - val_loss: 3.2304 - val_accuracy: 0.0409\n\nEpoch 00013: val_loss did not improve from 3.13228\n\nEpoch 00013: loss did not improve from 3.04834\nEpoch 14/1000\n26/26
[==============================] - 66s 3s/step - loss: 3.0549 - accuracy: 0.1477 - val_loss: 3.2015 - val_accuracy: 0.0409\n\nEpoch 00014: val_loss did not improve from 3.13228\n\nEpoch 00014: loss did not improve from 3.04834\nEpoch 15/1000\n26/26
[==============================] - 67s 3s/step - loss:

3.0755 – accuracy: 0.1370 – val_loss: 3.1619 –
val_accuracy: 0.0409\n\nEpoch 00015: val_loss did not
improve from 3.13228\n\nEpoch 00015: loss did not improve
from 3.04834\nEpoch 16/1000\n26/26
[==============================] – 66s 3s/step – loss:
3.0631 – accuracy: 0.1446 – val_loss: 3.1893 –
val_accuracy: 0.0409\n\nEpoch 00016: val_loss did not
improve from 3.13228\n\nEpoch 00016: loss did not improve
from 3.04834\nEpoch 17/1000\n26/26
[==============================] – 67s 3s/step – loss:
3.0558 – accuracy: 0.1496 – val_loss: 3.2430 –
val_accuracy: 0.0409\n\nEpoch 00017: val_loss did not
improve from 3.13228\n\nEpoch 00017: loss did not improve
from 3.04834\nEpoch 18/1000\n26/26
[==============================] – 61s 2s/step – loss:
3.0616 – accuracy: 0.1471 – val_loss: 3.1461 –
val_accuracy: 0.0409\n\nEpoch 00018: val_loss did not
improve from 3.13228\n\nEpoch 00018: loss did not improve
from 3.04834\nEpoch 19/1000\n26/26
[==============================] – 659s 25s/step – loss:
3.0645 – accuracy: 0.1420 – val_loss: 3.2351 –
val_accuracy: 0.0409\n\nEpoch 00019: val_loss did not
improve from 3.13228\n\nEpoch 00019: loss did not improve
from 3.04834\nEpoch 20/1000\n26/26
[==============================] – 54s 2s/step – loss:
3.0592 – accuracy: 0.1444 – val_loss: 3.1370 –
val_accuracy: 0.0409\n\nEpoch 00020: val_loss did not
improve from 3.13228\n\nEpoch 00020: loss did not improve
from 3.04834\nEpoch 21/1000\n26/26
[==============================] – 80s 3s/step – loss:
3.0646 – accuracy: 0.1472 – val_loss: 3.1770 –
val_accuracy: 0.0409\n\nEpoch 00021: val_loss did not
improve from 3.13228\n\nEpoch 00021: loss did not improve
from 3.04834\nEpoch 22/1000\n26/26
[==============================] – 64s 2s/step – loss:
3.0630 – accuracy: 0.1475 – val_loss: 3.2277 –
val_accuracy: 0.0409\n\nEpoch 00022: val_loss did not
improve from 3.13228\n\nEpoch 00022: loss did not improve
from 3.04834\nEpoch 23/1000\n26/26
[==============================] – 55s 2s/step – loss:
3.0635 – accuracy: 0.1408 – val_loss: 3.1488 –
val_accuracy: 0.0409\n\nEpoch 00023: val_loss did not
improve from 3.13228\n\nEpoch 00023: loss did not improve
from 3.04834\nEpoch 24/1000\n26/26
[==============================] – 70s 3s/step – loss:

3.0685 – accuracy: 0.1409 – val_loss: 3.1949 –
val_accuracy: 0.0409\n\nEpoch 00024: val_loss did not
improve from 3.13228\n\nEpoch 00024: loss did not improve
from 3.04834\nEpoch 25/1000\n26/26
[==============================] – 65s 3s/step – loss:
3.0628 – accuracy: 0.1472 – val_loss: 3.1538 –
val_accuracy: 0.0409\n\nEpoch 00025: val_loss did not
improve from 3.13228\n\nEpoch 00025: loss did not improve
from 3.04834\nEpoch 26/1000\n26/26
[==============================] – 66s 3s/step – loss:
3.0594 – accuracy: 0.1456 – val_loss: 3.2175 –
val_accuracy: 0.0409\n\nEpoch 00026: val_loss did not
improve from 3.13228\n\nEpoch 00026: loss did not improve
from 3.04834\nEpoch 27/1000\n26/26
[==============================] – 65s 2s/step – loss:
3.0626 – accuracy: 0.1444 – val_loss: 3.2237 –
val_accuracy: 0.0409\n\nEpoch 00027: val_loss did not
improve from 3.13228\n\nEpoch 00027: loss did not improve
from 3.04834\nEpoch 28/1000\n26/26
[==============================] – 63s 2s/step – loss:
3.0645 – accuracy: 0.1432 – val_loss: 3.2014 –
val_accuracy: 0.0409\n\nEpoch 00028: val_loss did not
improve from 3.13228\n\nEpoch 00028: loss did not improve
from 3.04834\nEpoch 29/1000\n26/26
[==============================] – 59s 2s/step – loss:
3.0645 – accuracy: 0.1438 – val_loss: 3.2313 –
val_accuracy: 0.0409\n\nEpoch 00029: val_loss did not
improve from 3.13228\n\nEpoch 00029: loss did not improve
from 3.04834\nEpoch 30/1000\n26/26
[==============================] – 54s 2s/step – loss:
3.0619 – accuracy: 0.1468 – val_loss: 3.2027 –
val_accuracy: 0.0409\n\nEpoch 00030: val_loss did not
improve from 3.13228\n\nEpoch 00030: loss did not improve
from 3.04834\nEpoch 31/1000\n26/26
[==============================] – 50s 2s/step – loss:
3.0614 – accuracy: 0.1438 – val_loss: 3.1087 –
val_accuracy: 0.0409\n\nEpoch 00031: val_loss improved
from 3.13228 to 3.10871, saving model to weight.
checkpoint5.cont.h5\n\nEpoch 00031: loss did not improve
from 3.04834\nEpoch 32/1000\n26/26
[==============================] – 51s 2s/step – loss:
3.0640 – accuracy: 0.1450 – val_loss: 3.1754 –
val_accuracy: 0.0409\n\nEpoch 00032: val_loss did not
improve from 3.10871\n\nEpoch 00032: loss did not improve
from 3.04834\nEpoch 33/1000\n26/26

```
    [==============================] - 51s 2s/step - loss:
    3.0563 - accuracy: 0.1475 - val_loss: 3.2294 -
    val_accuracy: 0.0409\n\nEpoch 00033: val_loss did not
    improve from 3.10871\n\nEpoch 00033: loss did not improve
    from 3.04834\nEpoch 34/1000\n"
},
{
 "name": "stdout",
 "output_type": "stream",
 "text": "26/26 [==============================] - 59s 2s/step
     - loss: 3.0678 - accuracy: 0.1408 - val_loss: 3.1755 -
    val_accuracy: 0.0409\n\nEpoch 00034: val_loss did not
    improve from 3.10871\n\nEpoch 00034: loss did not improve
    from 3.04834\nEpoch 35/1000\n26/26
    [==============================] - 54s 2s/step - loss:
    3.0665 - accuracy: 0.1481 - val_loss: 3.1536 -
    val_accuracy: 0.0409\n\nEpoch 00035: val_loss did not
    improve from 3.10871\n\nEpoch 00035: loss did not improve
    from 3.04834\nEpoch 36/1000\n26/26
    [==============================] - 57s 2s/step - loss:
    3.0547 - accuracy: 0.1514 - val_loss: 3.1829 -
    val_accuracy: 0.0409\n\nEpoch 00036: val_loss did not
    improve from 3.10871\n\nEpoch 00036: loss did not improve
    from 3.04834\nEpoch 37/1000\n26/26
    [==============================] - 65s 3s/step - loss:
    3.0693 - accuracy: 0.1347 - val_loss: 3.2082 -
    val_accuracy: 0.0409\n\nEpoch 00037: val_loss did not
    improve from 3.10871\n\nEpoch 00037: loss did not improve
    from 3.04834\nEpoch 38/1000\n26/26
    [==============================] - 65s 2s/step - loss:
    3.0557 - accuracy: 0.1477 - val_loss: 3.1935 -
    val_accuracy: 0.0409\n\nEpoch 00038: val_loss did not
    improve from 3.10871\n\nEpoch 00038: loss did not improve
    from 3.04834\nEpoch 39/1000\n26/26
    [==============================] - 58s 2s/step - loss:
    3.0518 - accuracy: 0.1562 - val_loss: 3.1678 -
    val_accuracy: 0.0409\n\nEpoch 00039: val_loss did not
    improve from 3.10871\n\nEpoch 00039: loss did not improve
    from 3.04834\nEpoch 40/1000\n26/26
    [==============================] - 56s 2s/step - loss:
    3.0759 - accuracy: 0.1329 - val_loss: 3.1427 -
    val_accuracy: 0.0409\n\nEpoch 00040: val_loss did not
    improve from 3.10871\n\nEpoch 00040: loss did not improve
    from 3.04834\nEpoch 41/1000\n26/26
    [==============================] - 59s 2s/step - loss:
```

```
    3.0762 - accuracy: 0.1348 - val_loss: 3.2024 -
    val_accuracy: 0.0409\n\nEpoch 00041: val_loss did not
    improve from 3.10871\n\nEpoch 00041: loss did not improve
    from 3.04834\nEpoch 42/1000\n26/26
    [==============================] - 54s 2s/step - loss:
    3.0533 - accuracy: 0.1541 - val_loss: 3.2151 -
    val_accuracy: 0.0409\n\nEpoch 00042: val_loss did not
    improve from 3.10871\n\nEpoch 00042: loss did not improve
    from 3.04834\nEpoch 43/1000\n26/26
    [==============================] - 53s 2s/step - loss:
    3.0579 - accuracy: 0.1460 - val_loss: 3.1284 -
    val_accuracy: 0.0409\n\nEpoch 00043: val_loss did not
    improve from 3.10871\n\nEpoch 00043: loss did not improve
    from 3.04834\nEpoch 44/1000\n10/26
    [==========>..................] - ETA: 33s - loss: 3.0735
     - accuracy: 0.1458"
},
{
 "ename": "KeyboardInterrupt",
 "evalue": "",
 "output_type": "error",
 "traceback": [
  "\u001b[0;31m
     ---------------------------------------------------------------------------
     u001b[0m",
  "\u001b[0;31mKeyboardInterrupt\u001b[0m
                           Traceback (most recent call last)
     ",
  "\u001b[0;32m<ipython-input-11-a42e6152a514>\u001b[0m in \
     u001b[0;36m<module>\u001b[0;34m()\u001b[0m\n\u001b[1;32m
          6\u001b[0m          \u001b[0mvalidation_data\u001b[0m
     \u001b[0;34m=\u001b[0m\u001b[0mtest_generator\u001b[0m\
     u001b[0;34m,\u001b[0m\u001b[0;34m\u001b[0m\u001b[0m\n\
     u001b[1;32m      7\u001b[0m          \u001b[0
     mvalidation_steps\u001b[0m\u001b[0;34m=\u001b[0m \u001b
     [0;36m2\u001b[0m\u001b[0;34m,\u001b[0m\u001b[0;34m\u001b
     [0m\u001b[0m\n\u001b[0;32m----> 8\u001b[0;31m
     callbacks=checkpoint)\n\u001b[0m",
  "\u001b[0;32m/Library/Frameworks/Python.framework/Versions
     /2.7/lib/python2.7/site-packages/keras/legacy/interfaces.
     pyc\u001b[0m in \u001b[0;36mwrapper\u001b[0;34m(*args, **
     kwargs)\u001b[0m\n\u001b[1;32m     89\u001b[0m
                  warnings.warn('Update your `' +
     object_name + '` call to the ' +\n\u001b[1;32m     90\
     u001b[0m                              'Keras 2 API: ' +
```

```
    signature, stacklevel=2)\n\u001b[0;32m----> 91\u001b[0;31m
                \u001b[0;32mreturn\u001b[0m \u001b[0mfunc\
    u001b[0m\u001b[0;34m(\u001b[0m\u001b[0;34m*\u001b[0m\
    u001b[0margs\u001b[0m\u001b[0;34m,\u001b[0m \u001b[0;34m
    **\u001b[0m\u001b[0mkwargs\u001b[0m\u001b[0;34m)\u001b[0m
    \u001b[0;34m\u001b[0m\u001b[0m\n\u001b[0m\u001b[1;32m
        92\u001b[0m           \u001b[0mwrapper\u001b[0m\u001b
    [0;34m.\u001b[0m\u001b[0m_original_function\u001b[0m \
    u001b[0;34m=\u001b[0m \u001b[0mfunc\u001b[0m\u001b[0;34m\
    u001b[0m\u001b[0m\n\u001b[1;32m      93\u001b[0m          \
    u001b[0;32mreturn\u001b[0m \u001b[0mwrapper\u001b[0m\
    u001b[0;34m\u001b[0m\u001b[0m\n",
"\u001b[0;32m/Library/Frameworks/Python.framework/Versions
    /2.7/lib/python2.7/site-packages/keras/engine/training.
    pyc\u001b[0m in \u001b[0;36mfit_generator\u001b[0;34m(
    self, generator, steps_per_epoch, epochs, verbose,
    callbacks, validation_data, validation_steps,
    validation_freq, class_weight, max_queue_size, workers,
    use_multiprocessing, shuffle, initial_epoch)\u001b[0m\n\
    u001b[1;32m   1730\u001b[0m              \u001b[0
    muse_multiprocessing\u001b[0m\u001b[0;34m=\u001b[0m\u001b
    [0muse_multiprocessing\u001b[0m\u001b[0;34m,\u001b[0m\
    u001b[0;34m\u001b[0m\u001b[0m\n\u001b[1;32m   1731\u001b
    [0m              \u001b[0mshuffle\u001b[0m\u001b[0;34m=\
    u001b[0m\u001b[0mshuffle\u001b[0m\u001b[0;34m,\u001b[0m\
    u001b[0;34m\u001b[0m\u001b[0m\n\u001b[0;32m-> 1732\u001b
    [0;31m              initial_epoch=initial_epoch)\n\u001b[0
    m\u001b[1;32m   1733\u001b[0m \u001b[0;34m\u001b[0m\u001b
    [0m\n\u001b[1;32m   1734\u001b[0m     \u001b[0;34m@\u001b
    [0m\u001b[0minterfaces\u001b[0m\u001b[0;34m.\u001b[0m\
    u001b[0mlegacy_generator_methods_support\u001b[0m\u001b
    [0;34m\u001b[0m\u001b[0m\n",
"\u001b[0;32m/Library/Frameworks/Python.framework/Versions
    /2.7/lib/python2.7/site-packages/keras/engine/
    training_generator.pyc\u001b[0m in \u001b[0;36
    mfit_generator\u001b[0;34m(model, generator,
    steps_per_epoch, epochs, verbose, callbacks,
    validation_data, validation_steps, validation_freq,
    class_weight, max_queue_size, workers,
    use_multiprocessing, shuffle, initial_epoch)\u001b[0m\n\
    u001b[1;32m    218\u001b[0m
                                                \u001b[0
    msample_weight\u001b[0m\u001b[0;34m=\u001b[0m\u001b[0
    msample_weight\u001b[0m\u001b[0;34m,\u001b[0m\u001b[0;34m
    \u001b[0m\u001b[0m\n\u001b[1;32m    219\u001b[0m
```

\u001b[0
mclass_weight\u001b[0m\u001b[0;34m=\u001b[0m\u001b[0
mclass_weight\u001b[0m\u001b[0;34m,\u001b[0m\u001b[0;34m\
u001b[0m\u001b[0m\n\u001b[0;32m—> 220\u001b[0;31m
reset_metrics
=False)\n\u001b[0m\u001b[1;32m 221\u001b[0m \u001b
[0;34m\u001b[0m\u001b[0m\n\u001b[1;32m 222\u001b[0m
\u001b[0mouts\u001b[0m \u001b[0;34m=\
u001b[0m \u001b[0mto_list\u001b[0m\u001b[0;34m(\u001b[0m\
u001b[0mouts\u001b[0m\u001b[0;34m)\u001b[0m\u001b[0;34m\
u001b[0m\u001b[0m\n",
"\u001b[0;32m/Library/Frameworks/Python.framework/Versions
/2.7/lib/python2.7/site-packages/keras/engine/training.
pyc\u001b[0m in \u001b[0;36mtrain_on_batch\u001b[0;34m(
self, x, y, sample_weight, class_weight, reset_metrics)\
u001b[0m\n\u001b[1;32m 1512\u001b[0m \u001b
[0mins\u001b[0m \u001b[0;34m=\u001b[0m \u001b[0mx\u001b[0
m \u001b[0;34m+\u001b[0m \u001b[0my\u001b[0m \u001b[0;34m
+\u001b[0m \u001b[0msample_weights\u001b[0m\u001b[0;34m\
u001b[0m\u001b[0m\n\u001b[1;32m 1513\u001b[0m \
u001b[0mself\u001b[0m\u001b[0;34m.\u001b[0m\u001b[0
m_make_train_function\u001b[0m\u001b[0;34m(\u001b[0m\
u001b[0;34m)\u001b[0m\u001b[0;34m\u001b[0m\u001b[0m\n\
u001b[0;32m—> 1514\u001b[0;31m \u001b[0moutputs\
u001b[0m \u001b[0;34m=\u001b[0m \u001b[0mself\u001b[0m\
u001b[0;34m.\u001b[0m\u001b[0mtrain_function\u001b[0m\
u001b[0;34m(\u001b[0m\u001b[0mins\u001b[0m\u001b[0;34m)\
u001b[0m\u001b[0;34m\u001b[0m\u001b[0m\n\u001b[0m\u001b
[1;32m 1515\u001b[0m \u001b[0;34m\u001b[0m\u001b[0m\n\
u001b[1;32m 1516\u001b[0m \u001b[0;32mif\u001b
[0m \u001b[0mreset_metrics\u001b[0m\u001b[0;34m:\u001b[0m
\u001b[0;34m\u001b[0m\u001b[0m\n",
"\u001b[0;32m/Library/Frameworks/Python.framework/Versions
/2.7/lib/python2.7/site-packages/tensorflow_core/python/
keras/backend.pyc\u001b[0m in \u001b[0;36m__call__\u001b
[0;34m(self, inputs)\u001b[0m\n\u001b[1;32m 3725\u001b
[0m \u001b[0mvalue\u001b[0m \u001b[0;34m=\u001b[0
m \u001b[0mmath_ops\u001b[0m\u001b[0;34m.\u001b[0m\u001b
[0mcast\u001b[0m\u001b[0;34m(\u001b[0m\u001b[0mvalue\
u001b[0m\u001b[0;34m,\u001b[0m \u001b[0mtensor\u001b[0m\
u001b[0;34m.\u001b[0m\u001b[0mdtype\u001b[0m\u001b[0;34m)
\u001b[0m\u001b[0;34m\u001b[0m\u001b[0m\n\u001b[1;32m
3726\u001b[0m \u001b[0mconverted_inputs\u001b[0m\
u001b[0;34m.\u001b[0m\u001b[0mappend\u001b[0m\u001b[0;34m
(\u001b[0m\u001b[0mvalue\u001b[0m\u001b[0;34m)\u001b[0m\

u001b[0;34m\u001b[0m\u001b[0m\n\u001b[0;32m–> 3727\u001b[0;31m \u001b[0moutputs\u001b[0m \u001b[0;34m=\u001b[0m \u001b[0mself\u001b[0m\u001b[0;34m.\u001b[0m\u001b[0m_graph_fn\u001b[0m\u001b[0;34m(\u001b[0m\u001b[0;34m*\u001b[0m\u001b[0mconverted_inputs\u001b[0m\u001b[0;34m)\u001b[0m\u001b[0;34m\u001b[0m\u001b[0m\n\u001b[0m\u001b[1;32m 3728\u001b[0m \u001b[0;34m\u001b[0m\u001b[0m\n\u001b[1;32m 3729\u001b[0m \u001b[0;31m# EagerTensor.numpy() will often make a copy to ensure memory safety.\u001b[0m\u001b[0;34m\u001b[0m\u001b[0;34m\u001b[0m\u001b[0m\n"

"\u001b[0;32m/Library/Frameworks/Python.framework/Versions/2.7/lib/python2.7/site–packages/tensorflow_core/python/eager/function.pyc\u001b[0m in \u001b[0;36m__call__\u001b[0;34m(self, *args, **kwargs)\u001b[0m\n\u001b[1;32m 1549\u001b[0m \u001b[0mTypeError\u001b[0m\u001b[0;34m:\u001b[0m \u001b[0mFor\u001b[0m \u001b[0minvalid\u001b[0m \u001b[0mpositional\u001b[0m\u001b[0;34m/\u001b[0m\u001b[0mkeyword\u001b[0m \u001b[0margument\u001b[0m \u001b[0mcombinations\u001b[0m\u001b[0;34m.\u001b[0m\u001b[0;34m\u001b[0m\u001b[0m\n\u001b[0;34m\u001b[0m\u001b[0m\n\u001b[1;32m 1550\u001b[0m \"\"\"\n\u001b[0;32m–> 1551\u001b[0;31m \u001b[0;32mreturn\u001b[0m \u001b[0mself\u001b[0m\u001b[0;34m.\u001b[0m\u001b[0m_call_impl\u001b[0m\u001b[0;34m(\u001b[0m\u001b[0margs\u001b[0m\u001b[0;34m,\u001b[0m \u001b[0mkwargs\u001b[0m\u001b[0;34m)\u001b[0m\u001b[0;34m\u001b[0m\u001b[0;34m\u001b[0m\u001b[0m\n\u001b[0m\u001b[1;32m 1552\u001b[0m \u001b[0;34m\u001b[0m\u001b[0m\n\u001b[1;32m 1553\u001b[0m \u001b[0;32mdef\u001b[0m \u001b[0m_call_impl\u001b[0m\u001b[0;34m(\u001b[0m\u001b[0mself\u001b[0m\u001b[0;34m,\u001b[0m \u001b[0margs\u001b[0m\u001b[0;34m,\u001b[0m \u001b[0mkwargs\u001b[0m\u001b[0;34m,\u001b[0m \u001b[0mcancellation_manager\u001b[0m\u001b[0;34m=\u001b[0m\u001b[0;32mNone\u001b[0m\u001b[0;34m)\u001b[0m\u001b[0;34m:\u001b[0m\u001b[0;34m\u001b[0m\u001b[0;34m\u001b[0m\u001b[0m\n"

"\u001b[0;32m/Library/Frameworks/Python.framework/Versions/2.7/lib/python2.7/site–packages/tensorflow_core/python/eager/function.pyc\u001b[0m in \u001b[0;36m_call_impl\u001b[0;34m(self, args, kwargs, cancellation_manager)\u001b[0m\n\u001b[1;32m 1589\u001b[0m raise TypeError(\"Keyword arguments {} unknown. Expected {}.\".format(\n\u001b[1;32m 1590\u001b[0m list(kwargs.keys()), list(self._arg_keywords)))\n\u001b[0;32m–> 1591\u001b[0;31m \u001b[0;32mreturn\u001b[0m \u001b[0mself\u001b[0m\u001b[0;34m.\u001b[0m\u001b[0m

```
    m_call_flat\u001b[0m\u001b[0;34m(\u001b[0m\u001b[0margs\
    u001b[0m\u001b[0;34m,\u001b[0m \u001b[0mself\u001b[0m\
    u001b[0;34m.\u001b[0m\u001b[0mcaptured_inputs\u001b[0m\
    u001b[0;34m,\u001b[0m \u001b[0mcancellation_manager\u001b
    [0m\u001b[0;34m)\u001b[0m\u001b[0;34m\u001b[0m\u001b[0m\n
    \u001b[0m\u001b[1;32m  1592\u001b[0m \u001b[0;34m\u001b
    [0m\u001b[0m\n\u001b[1;32m  1593\u001b[0m  \u001b[0;32
    mdef\u001b[0m \u001b[0m_filtered_call\u001b[0m\u001b[0;34
    m(\u001b[0m\u001b[0mself\u001b[0m\u001b[0;34m,\u001b[0m \
    u001b[0margs\u001b[0m\u001b[0;34m,\u001b[0m \u001b[0
    mkwargs\u001b[0m\u001b[0;34m)\u001b[0m\u001b[0;34m:\u001b
    [0m\u001b[0;34m\u001b[0m\u001b[0m\n",
"\u001b[0;32m/Library/Frameworks/Python.framework/Versions
    /2.7/lib/python2.7/site-packages/tensorflow_core/python/
    eager/function.pyc\u001b[0m in \u001b[0;36m_call_flat\
    u001b[0;34m(self, args, captured_inputs,
    cancellation_manager)\u001b[0m\n\u001b[1;32m  1690\u001b
    [0m        \u001b[0;31m# No tape is watching; skip to
    running the function.\u001b[0m\u001b[0;34m\u001b[0m\u001b
    [0;34m\u001b[0m\u001b[0m\n\u001b[1;32m  1691\u001b[0m
          return self._build_call_outputs(self.
    _inference_function.call(\n\u001b[0;32m-> 1692\u001b[0;31
    m            ctx, args, cancellation_manager=
    cancellation_manager))\n\u001b[0m\u001b[1;32m  1693\
    u001b[0m      forward_backward = self.
    _select_forward_and_backward_functions(\n\u001b[1;32m
    1694\u001b[0m          \u001b[0margs\u001b[0m\u001b[0;34m
    ,\u001b[0m\u001b[0;34m\u001b[0m\u001b[0m\n",
"\u001b[0;32m/Library/Frameworks/Python.framework/Versions
    /2.7/lib/python2.7/site-packages/tensorflow_core/python/
    eager/function.pyc\u001b[0m in \u001b[0;36mcall\u001b
    [0;34m(self, ctx, args, cancellation_manager)\u001b[0m\n\
    u001b[1;32m   543\u001b[0m                \u001b[0minputs
    \u001b[0m\u001b[0;34m=\u001b[0m\u001b[0margs\u001b[0m\
    u001b[0;34m,\u001b[0m\u001b[0;34m\u001b[0m\u001b[0m\n\
    u001b[1;32m   544\u001b[0m                \u001b[0mattrs\
    u001b[0m\u001b[0;34m=\u001b[0m\u001b[0;34m(\u001b[0m\
    u001b[0;34m\"executor_type\"\u001b[0m\u001b[0;34m,\u001b
    [0m \u001b[0mexecutor_type\u001b[0m\u001b[0;34m,\u001b[0m
     \u001b[0;34m\"config_proto\"\u001b[0m\u001b[0;34m,\u001b
    [0m \u001b[0mconfig\u001b[0m\u001b[0;34m)\u001b[0m\u001b
    [0;34m,\u001b[0m\u001b[0;34m\u001b[0m\u001b[0m\n\u001b
    [0;32m--> 545\u001b[0;31m                ctx=ctx)\n\u001b
    [0m\u001b[1;32m   546\u001b[0m          \u001b[0;32melse\
    u001b[0m\u001b[0;34m:\u001b[0m\u001b[0;34m\u001b[0m\u001b
```

```
        [0m\n\u001b[1;32m    547\u001b[0m          outputs =
        execute.execute_with_cancellation(\n",
      "\u001b[0;32m/Library/Frameworks/Python.framework/Versions
        /2.7/lib/python2.7/site-packages/tensorflow_core/python/
        eager/execute.pyc\u001b[0m in \u001b[0;36mquick_execute\
        u001b[0;34m(op_name, num_outputs, inputs, attrs, ctx,
        name)\u001b[0m\n\u001b[1;32m     59\u001b[0m     tensors
        = pywrap_tensorflow.TFE_Py_Execute(ctx._handle,
        device_name,\n\u001b[1;32m     60\u001b[0m
                                                \u001b[0
        mop_name\u001b[0m\u001b[0;34m,\u001b[0m \u001b[0minputs\
        u001b[0m\u001b[0;34m,\u001b[0m \u001b[0mattrs\u001b[0m\
        u001b[0;34m,\u001b[0m\u001b[0;34m\u001b[0m\u001b[0m\n\
        u001b[0;32m----> 61\u001b[0;31m

        num_outputs)\n\u001b[0m\u001b[1;32m     62\u001b[0m   \
        u001b[0;32mexcept\u001b[0m \u001b[0mcore\u001b[0m\u001b
        [0;34m.\u001b[0m\u001b[0m_NotOkStatusException\u001b[0m \
        u001b[0;32mas\u001b[0m \u001b[0me\u001b[0m\u001b[0;34m:\
        u001b[0m\u001b[0;34m\u001b[0m\u001b[0m\n\u001b[1;32m
        63\u001b[0m     \u001b[0;32mif\u001b[0m \u001b[0mname\
        u001b[0m \u001b[0;32mis\u001b[0m \u001b[0;32mnot\u001b[0m
         \u001b[0mNone\u001b[0m\u001b[0;34m:\u001b[0m\u001b[0;34m
        \u001b[0m\u001b[0m\n",
      "\u001b[0;31mKeyboardInterrupt\u001b[0m: "
     ]
    }
   ],
   "source": "hist = model.fit_generator(\n        train_generator
     ,\n        steps_per_epoch= len(train_generator.filenames)
     //64,\n        epochs=1000,\n        shuffle = True,\n
           validation_data=test_generator,\n
     validation_steps= 2,\n        callbacks=checkpoint)"
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {},
   "outputs": [],
   "source": ""
  }
 ],
 "metadata": {
  "kernelspec": {
   "display_name": "Python 2",
```

```
   "language": "python",
   "name": "python2"
  },
  "language_info": {
   "codemirror_mode": {
    "name": "ipython",
    "version": 2
   },
   "file_extension": ".py",
   "mimetype": "text/x-python",
   "name": "python",
   "nbconvert_exporter": "python",
   "pygments_lexer": "ipython2",
   "version": "2.7.16"
  }
 },
 "nbformat": 4,
 "nbformat_minor": 4
}
```

Lampiran 2

Source Code 2