SKRIPSI

IMPLEMENTASI KOMPUTASI PARALEL PADA APLIKASI *DEEP CONVOLUTIONAL NEURAL NETWORK* (STUDI KASUS : APLIKASI YUV *VIDEO SUPER-RESOLUTION*)

Disusun dan diajukan oleh:

RYAN TERRY THAHIR
D121181330

PROGRAM STUDI SARJANA TEKNIK INFORMATIKA
FAKULTAS TEKNIK
UNIVERSITAS HASANUDDIN
GOWA
2024

1

LEMBAR PENGESAHAN SKRIPSI

IMPLEMENTASI KOMPUTASI PARALEL PADA APLIKASI *DEEP CONVOLUTIONAL NEURAL NETWORK* (STUDI KASUS : APLIKASI *YUV VIDEO SUPER-RESOLUTION*)

Disusun dan diajukan oleh

Ryan Terry Thahir
D121181330

Telah dipertahankan di hadapan Panitia Ujian yang dibentuk dalam rangka Penyelesaian Studi Program Sarjana Program Studi Teknik Informatika
Fakultas Teknik Universitas Hasanuddin
Pada tanggal 2 Oktober 2024
dan dinyatakan telah memenuhi syarat kelulusan

Menyetujui,

Dr. Adnan, S.T., M.T,	Muhammad Alief Fahdal Imran Oemar, ST., M.Sc,
NIP 19740426 200501 1 002	NIP 19940522 202101 5 001

Prof. Dr. Ir. Indrabayu, ST., MT., M.Bus.Sys., IPM, ASEAN. Eng.,

NIP. 19750716 200212 1 004

2

PERNYATAAN KEASLIAN

Yang bertanda tangan dibawah ini ;

Nama	:	Ryan Terry Thahir
NIM	:	D121181330
Program Studi	:	Teknik Informatika
Jenjang	:	S1

Menyatakan dengan ini bahwa karya tulisan saya berjudul

Implementasi Komputasi Paralel pada Aplikasi *Deep Convolutional Neural Network* (Studi Kasus : Aplikasi *YUV Video Super-Resolution*)

Adalah karya tulisan saya sendiri dan bukan merupakan pengambilan alihan tulisan orang lain dan bahwa skripsi yang saya tulis ini benar-benar merupakan hasil karya saya sendiri.

Semua informasi yang ditulis dalam skripsi yang berasal dari penulis lain telah diberi penghargaan, yakni dengan mengutip sumber dan tahun penerbitannya. Oleh karena itu semua tulisan dalam skripsi ini sepenuhnya menjadi tanggung jawab penulis. Apabila ada pihak manapun yang merasa ada kesamaan judul dan atau hasil temuan dalam skripsi ini, maka penulis siap untuk diklarifikasi dan mempertanggungjawabkan segala resiko.

Segala data dan informasi yang diperoleh selama proses pembuatan skripsi, yang akan dipublikasi oleh Penulis di masa depan harus mendapat persetujuan dari Dosen Pembimbing.

Apabila dikemudian hari terbukti atau dapat dibuktikan bahwa sebagian atau keseluruhan isi skripsi ini hasil karya orang lain, maka saya bersedia menerima sanksi atas perbuatan tersebut.

Gowa, 2 Oktober 2024

Yang Menyatakan

Ryan Terry Thahir

ABSTRAK

RYAN TERRY THAHIR. *Implementasi Komputasi Paralel pada Aplikasi Deep Convolutional Neural Network (Studi Kasus : Aplikasi YUV Video Super-Resolution)* (dibimbing oleh Dr. Adnan, S.T. M.T dan Muhammad Alief Fahdal Imran Oemar, ST., M.Sc)

Aplikasi YUV *Video Super-Resolution* merupakan aplikasi yang digunakan untuk meningkatkan resolusi video dengan format YUV sebesar 2 kali lipat dengan menggunakan salah satu bentuk *DCNN* (*Deep Convolutional Neural Network*) yaitu algoritma *FSRCNN* (*Fast Super Resolution Convolutional Neural Network*), namun waktu yang dibutuhkan aplikasi untuk mengolah sebuah video yang terdiri atas 150 *frame* adalah 800 detik atau sekitar 13 menit. Untuk mengatasi waktu eksekusi yang lama dapat digunakan komputasi paralel untuk mengurangi waktu eksekusi aplikasi. Bentuk implementasi komputasi paralel yang digunakan adalah *multithreading* dari *library OpenMP*. Untuk mengukur performa aplikasi dengan implementasi komputasi paralel digunakan *CPU time* dan *wall time* dengan menggunakan *input* video yang memiliki jumlah *frame* yang berbeda. Data *CPU time* dan *wall time* dapat digunakan untuk mendapatkan nilai *speedup* tiap video di masing -masing aplikasi paralel. Nilai *speedup* digunakan untuk membandingkan antar aplikasi paralel dengan tingkat paralelisasi blok kode yang berbeda. Setelah itu, *speedup* aktual dibandingkan dengan *speedup* teoritis yang didapatkan dengan menggunakan hukum Amdahl. Selanjutnya untuk memastikan kualitas gambar dari video yang dihasilkan oleh aplikasi paralel, video dapat diamati secara langsung dan juga metrik *PSNR* (*Peak Signal-to-Noise Ratio*). Hasil yang didapatkan melalui penelitian ini adalah blok kode yang diparalelkan untuk mendapatkan nilai *speedup* tertinggi (waktu eksekusi terendah) untuk seluruh video yang diolah dicapai oleh program dengan implementasi komputasi paralel pada *layer* 1, *layer* 2, *layer* 3, *layer* 4, *layer* 5, *layer* 6, dan *layer* 7
Kata Kunci: Aplikasi YUV Video Super-Resolution, komputasi paralel, konvolusi, DCNN (*Deep Convolutional Neural Network*), FSRCNN (Fast Super Resolution Convolutional Neural Network), dekonvolusi, *multithreading*, *OpenMP*, hukum Amdahl, *PSNR* (Peak Signal-to-Noise Ratio).

ABSTRACT

RYAN TERRY THAHIR. *Implementing Parallel Computation to Deep Convolutional Neural Network (Case Study : YUV Video Super-Resolution Application)* (supervised by Dr. Adnan, S.T. M.T and Muhammad Alief Fahdal Imran Oemar, ST., M.Sc)

YUV Video Super-Resolution app is an app used to increase the resolution of a video with YUV video format by two times by using one of DCNN (Deep Convolutional Neural Network) form which is FSRCNN (Fast Super Resolution Convolutional Neural Network), but the time needed to process a 150 frames video is around 800 seconds or around 13 minutes. To overcome this long execution time, parallel computing can be used to decrease the execution time. One of the forms of parallel computing that could be used is multithreading from the OpenMP library. To measure the performance of the application with parallel computing implementation, CPU time and wall time is measured using input videos with different numbers of frames, CPU time and wall time data is then used to get the speedup value of each video of different parallel application. The speedup value then is used to compare between parallel applications with a different level of parallelization blocks of code. After that, the actual speedup then is used to compare with theoretical speedup acquired using Amdahl's law. Subsequently, to make sure the videos retain the quality, the video could be observed directly accompanied by using PSNR (Peak Signal-to- Noise Ratio) metric. Result of this research are blocks of codes that are parallelized to achieve the highest speedup are layer 1, layer 2, layer 3, layer 4, layer 5, layer 6, and layer 7.

Keywords: YUV Video Super-Resolution application, parallel computing, convolution, DCNN (Deep Convolutional Neural Network), FSRCNN (Fast Super Resolution Convolutional Neural Network), deconvolution, multithreading, OpenMP, Amdahl's law, PSNR (Peak Signal-to-Noise Ratio).

DAFTAR ISI

LEMBAR PENGESAHAN SKRIPSI..... 1
PERNYATAAN KEASLIAN.....2
ABSTRAK..... 3
ABSTRACT.....4
DAFTAR ISI.....5
DAFTAR GAMBAR.....7
DAFTAR TABEL.....8
DAFTAR SINGKATAN DAN ARTI SIMBOL..... 9
DAFTAR LAMPIRAN..... 10
KATA PENGANTAR.....11
BAB I
PENDAHULUAN.....1
1.1 Latar Belakang..... 1
1.2 Rumusan Masalah.....2
1.3 Tujuan Penelitian/Perancangan.....2
1.4 Manfaat Penelitian/Perancangan.....2
1.5 Batasan Masalah.....3
BAB II
TINJAUAN PUSTAKA..... 5
2.1 Komputasi Parallel.....5
2.2 OpenMP..... 6
2.2.1 Directives (Direktif).....7
2.2.2 Clause (Klausa).....8
2.3 Multicore Processor..... 15
2.4 Aplikasi YUV Video Super-Resolution.....16
2.4.1 Format Video YUV.....17
2.4.2 Cara Kerja Aplikasi / Alur Kerja Aplikasi.....17
2.5 Artificial Neural Network (ANN).....22
2.6 Convolutional Neural Network (CNN) dan Deep Convolutional Neural Network (DCNN).....23
2.7 Super-Resolution Convolutional Neural Network (SRCNN) dan Fast Super-Resolution Convolutional Neural Network (FSRCNN).....25
2.8 Peak Signal-to-Noise Ratio (PSNR)..... 29
2.9 Hukum Amdahl.....30
2.10 SSIM (Structural Similarity Index Measure)..... 31
BAB III
METODE PENELITIAN..... 33
3.1 Waktu dan Lokasi Penelitian.....33

3.2 Tahapan Penelitian..... 33
3.3 Instrumen Penelitian.....34
3.3.1 Perangkat keras yang digunakan.....34
3.3.2 Perangkat lunak yang digunakan..... 34
3.4 Usulan Sistem.....36
3.4.1 Flowchart Aplikasi Serial.....36
3.4.2 Rancangan Flowchart Aplikasi Paralel..... 38
3.5 Implementasi Komputasi Paralel..... 40
3.5.1 Implementasi Komputasi Paralel pada layer 1 FSRCNN..... 40
3.5.2 Implementasi Komputasi Paralel pada layer 2 sampai 7 FSRCNN..41
3.6 Variabel yang Diamati.....43
3.6.1 Waktu Eksekusi Aplikasi..... 43
3.6.2 Kualitas Gambar Video..... 44
3.6.3 Speedup..... 44
BAB IV
HASIL DAN PEMBAHASAN..... 47
4.1 Hasil Video Output dan Kualitas Gambar Video..... 47
4.2 Speedup Teoritis dan Speedup Aktual..... 53
4.2.1 Speedup Teoritis (Hukum Amdahl)..... 53
4.2.2 Speedup Aktual..... 55
BAB V
KESIMPULAN DAN SARAN..... 64
5.1 Kesimpulan..... 64
5.2 Saran.....64
DAFTAR PUSTAKA.....66

DAFTAR GAMBAR

Gambar 2.1 Ilustrasi pembagian tugas paralel..... 7
Gambar 2.2 Bagan konstruk OpenMP..... 7
Gambar 2.3 Hasil eksekusi static scheduling..... 11
Gambar 2.4 Hasil eksekusi dynamic scheduling.....12
Gambar 2.5 Hasil eksekusi dynamic scheduling dengan chunk size 3..... 13
Gambar 2.6 Hasil eksekusi guided scheduling..... 13
Gambar 2.7 Diagram Blok single core processor dan dual core processor (multicore processor)..... 15
Gambar 2.8 Ilustrasi komponen penyusun video format YUV.....17
Gambar 2.9 Video YUV sebagai file biner..... 18
Gambar 2.10 Ilustrasi proses konvolusi..... 19
Gambar 2.11 Proses konvolusi dalam ruang vektor.....20
Gambar 2.12 Ilustrasi proses dekonvolusi..... 21
Gambar 2.13 Artificial Neural Network..... 23
Gambar 2.14 Ilustrasi algoritma SRCNN dan FSRCNN.....26
Gambar 3.1 Ilustrasi tahapan penelitian.....33
Gambar 3.2 Flowchart fungsi main.....36
Gambar 3.3 Flowchart fungsi FSRCNN..... 37
Gambar 3.4 Flowchart layer k.....39
Gambar 3.5 Hasil implementasi kode untuk menampilkan CPU time dan wall time 44
Gambar 3.6 Output aplikasi ffmpeg untuk memperoleh nilai PSNR.....44
Gambar 4.1 Perbandingan video input dan video output aplikasi paralel hingga layer 7.....47
Gambar 4.2 Chart waktu eksekusi untuk video berjudul Suzie (150 frame)..... 60
Gambar 4.3 Chart waktu eksekusi untuk video berjudul Hall (300 frame)..... 61
Gambar 4.4 Chart waktu eksekusi untuk video berjudul Claire (494 frame)..... 61
Gambar 4.5 Chart waktu eksekusi untuk video berjudul Grandma (870 frame)... 62
Gambar 4.6 Chart waktu eksekusi untuk video berjudul Highway (2000 frame). 62

DAFTAR TABEL

Tabel 4.1 Data PSNR hasil perbandingan program serial dengan implementasi komputasi paralel pada layer 1.....48
Tabel 4.2 Data PSNR hasil perbandingan program serial dengan implementasi komputasi paralel pada layer 1 dan layer 2.....48
Tabel 4.3 Data PSNR hasil perbandingan program serial dengan implementasi komputasi paralel pada layer 1, layer 2, dan layer 3.....49
Tabel 4.4 Data PSNR hasil perbandingan program serial dengan implementasi komputasi paralel pada layer 1, layer 2, layer 3, dan layer 4.....50
Tabel 4.5 Data PSNR hasil perbandingan program serial dengan implementasi komputasi paralel pada layer 1, layer 2, layer 3, layer 4, dan layer 5.....51
Tabel 4.6 Data PSNR hasil perbandingan program serial dengan implementasi komputasi paralel pada layer 1, layer 2, layer 3, layer 4, layer 5, dan layer 6.....51
Tabel 4.7 Data PSNR hasil perbandingan program serial dengan implementasi komputasi paralel pada layer 1, layer 2, layer 3, layer 4, layer 5, layer 6, dan layer 7.....52
Tabel 4.8 Data SSIM perbandingan hasil output program serial dengan implementasi komputasi paralel untuk video berjudul Suzie..... 53
Tabel 4.9 Data untuk menghitung speedup berdasarkan hukum Amdahl.....54
Tabel 4.10 Speedup teoritis berdasarkan hukum Amdahl.....55
Tabel 4.11 Data waktu eksekusi program serial.....55
Tabel 4.12 Data waktu eksekusi program keseluruhan dengan implementasi komputasi paralel pada layer 1.....56
Tabel 4.13 Data waktu eksekusi program implementasi komputasi paralel pada pada layer 1 dan layer 2..... 56
Tabel 4.14 Data waktu eksekusi program implementasi komputasi paralel pada layer 1, layer 2, dan layer 3.....57
Tabel 4.15 Data waktu eksekusi program implementasi komputasi paralel pada layer 1, layer 2, layer 3, dan layer 4.....58
Tabel 4.16 Data waktu eksekusi program implementasi komputasi paralel pada layer 1, layer 2, layer 3, layer 4, dan layer 5.....58
Tabel 4.17 Data waktu eksekusi program implementasi komputasi paralel pada layer 1, layer 2, layer 3, layer 4, layer 5, dan layer 6.....59
Tabel 4.18 Data waktu eksekusi program implementasi komputasi paralel pada layer 1, layer 2, layer 3, layer 4, layer 5, layer 6, dan layer 7.....59

DAFTAR SINGKATAN DAN ARTI SIMBOL

Lambang/Singkatan	Arti dan Keterangan
CNN	*Convolutional Neural Network*
ANN	*Artificial Neural Network*
DCNN	*Deep Convolutional Neural Network*
SRCNN	*Super Resolution Convolutional Neural Network*
FSRCNN	*Fast Super Resolution Convolutional Neural Network*
LR	*Low Resolution*
HR	*High Resolution*
PSNR	*Peak Signal-to-Noise Ratio*

DAFTAR LAMPIRAN

Lampiran 1 Source code Program Serial YUV Video Super-Resolution..... 83
Lampiran 2 Source code Program Paralel YUV Video Super-Resolution.....121

KATA PENGANTAR

Puji dan syukur senantiasa kita panjatkan kehadirat Tuhan Yang Maha Esa karena atas berkat dan rahmat-Nya penulis dapat menyelesaikan skripsi dengan judul "Implementasi Komputasi Paralel pada Aplikasi *Deep Convolutional Neural Network* (Studi Kasus : Aplikasi *YUV Video Super-Resolution*)" yang merupakan salah satu syarat untuk memperoleh gelar sarjana pada Departemen Teknik Informatika Fakultas Teknik Universitas Hasanuddin. Selama proses pengerjaan skripsi ini penulis menerima begitu banyak bantuan dari berbagai pihak. Untuk itu peneliti memberikan ucapan terima kasih kepada :

1. Kepada ibu saya tercinta, Melinda Go yang selalu sabar dalam mendidik saya, mendoakan, memberikan semangat dan support selama proses penulisan skripsi dan selama masa studi.
2. Kakak dan adik saya , Felisia dan Aiko, yang selalu ada dan mendukung selama proses perkuliahan.
3. Bapak Prof. Dr. Ir. Indrabayu., S.T., M.T., M.Bus.Sys., IPM, ASEAN.Eng. selaku Ketua Departemen Teknik Informatika Fakultas Teknik Universitas Hasanuddin yang senantiasa memberikan bimbingan selama masa perkuliahan penulis.
4. Bapak Adnan, S.T., M.T., Ph.D. selaku pembimbing pertama dan Bapak Muhammad Alief Fahdal Imran Oemar, ST., M.Sc. selaku pembimbing kedua yang senantiasa memberikan waktu, tenaga dan pikiran serta perhatian yang sangat luar biasa dalam pembimbing penulis selama penyusunan Tugas Akhir ini.
5. Bapak Dr. Eng. Muhammad Niswar, S.T., M.IT. selaku pembimbing akademik yang senantiasa meluangkan waktu selama masa studi penulis.
6. Teman-teman dari kelas B yang telah membantu penulis sejak awal perkuliahan dan selalu membantu dalam penyelesaian Tugas Akhir.
7. Teman-teman SYNCHRONOUS18 yang telah membantu penulis sejak awal perkuliahan dan selalu membantu dalam penyelesaian Tugas Akhir.
8. Kakak-kakak senior yang senantiasa memberikan bantuan, dukungan dan semangat selama di Lab IoT and Parallel Computing.
9. Rahmadani dan Putri Alisyah selalu memberikan semangat dan bantuan selama pengerjaan skripsi.
10. Rachmat Maulana Nur dan Salahuddin yang selalu memberikan bantuan selama proses perkuliahan.
11. Malvin yang telah membantu saya perhitungan manual untuk modifikasi kode, menemukan penyebab masalah dalam salah satu implementasi kode, dan untuk selalu menjadi teman main game di waktu luang.
12. Jerry, Lorens, Anglie, Wanta, Derick, Henry, Eve, Bustan, Meidy, Enrico, dan Ko Novel yang selalu menghibur dan menjadi pendengar curahan hati yang sangat baik selama proses penulisan skripsi.
13. Keluarga Mahasiswa Katolik Teknik Universitas Hasanuddin (KMKT-UH) yang menjadi salah satu tempat belajar dan bersosialisasi, khususnya

teman-teman RISE UP 2018 yang telah membantu penulis selama masa perkuliahan.

14. Semua orang yang penulis belum sempat penulis sebutkan, terima kasih atas doa dan bantuan selama ini.

Penulis menyadari bahwa skripsi ini masih jauh dari sempurna walaupun telah menerima bantuan dari berbagai pihak. Mohon maaf apabila terdapat kesalahan dalam skripsi ini, oleh karena itu penulis menerima segala bentuk kritik dan saran yang dapat membangun dan menyempurnakan skripsi ini. Semoga skripsi ini dapat bermanfaat bagi banyak orang.

Gowa, Oktober 2024

Penulis

BAB I
PENDAHULUAN

1.1 Latar Belakang

Teknologi pemrosesan citra dan video mengalami kemajuan yang pesat beberapa tahun ini, adapun salah satu faktor pendorong kemajuan ini adalah pengembangan algoritma *deep learning*. Salah satu algoritma *deep learnin*g yang sangat populer dalam pengolahan citra yaitu *deep convolutional neural network (DCNN/CNN)*. *Deep convolutional neural network* merupakan salah satu tipe *artificial neural network (ANN)* yang pada umumnya digunakan untuk mengolah citra dan video, dimana *DCNN* terdiri atas beberapa *convolution layer*. *Deep convolutional neural network* memiliki cakupan penggunaan yang luas mulai dari klasifikasi gambar, text mining, visual recognition, deteksi objek, hingga peningkatan resolusi video. (Alzubaidi, 2021).

Namun tentunya *deep convolutional neural network* masih memiliki beberapa batasan, salah satu batasan atau masalah yang seringkali dijumpai adalah waktu komputasi yang dibutuhkan untuk memproses data *input* sangat lama. Sebagai contoh nyata dari batasan ini adalah masalah waktu komputasi aplikasi *YUV Video Super-Resolution*. Aplikasi *YUV Video Super-Resolution* merupakan aplikasi open source yang menggunakan algoritma *Fast Super-Resolution Convolutional Neural Networks* / *FSRCNN* untuk memetakan *input* video dengan format file *YUV* yang awalnya *Low Resolution*/*LR* (resolusi rendah) menjadi video *High Resolution*/HR (resolusi tinggi). Aplikasi membutuhkan waktu sekitar 800 detik atau kurang lebih 13 menit untuk memproses video *YUV* yang hanya terdiri atas 150 *frame* dimana spesifikasi perangkat yang digunakan yaitu Laptop Asus Model Zenbook UM431DA dengan prosesor AMD Ryzen 7 3700U (4 *Cores*/8 *Threads*, 6MB *cache*) dengan Radeon Vega Mobile Gfx 2.30 GHz dan *random access memory* (RAM) sebesar 8.00 GB tipe DDR4.

Untuk mengatasi masalah tersebut salah satu pendekatan yang dapat digunakan yaitu komputasi paralel. Komputasi paralel merupakan konsep dimana komputasi/proses dilakukan secara paralel/bersamaan oleh beberapa *processing unit*, dengan kata lain proses yang biasanya dijalankan secara sekuensial dapat

langsung berjalan bersamaan sehingga mengurangi waktu yang dibutuhkan untuk mengeksekusi program dan meningkatkan performa aplikasi. Salah satu konsep dari komputasi paralel ini adalah *multithreading*. *Multithreading* adalah konsep pemrograman dimana urutan instruksi dalam sebuah program yang dieksekusi secara konkuren/bersamaan oleh masing-masing *thread*, dengan demikian dapat meningkatkan performa serta mengutilisasi sumber daya komputasi dengan efisien dan lebih baik. Oleh karena itu, dengan mengimplementasikan komputasi paralel (*multithreading*) pada aplikasi *YUV Video Super-Resolution* waktu yang dibutuhkan untuk menghasilkan video diharapkan dapat berkurang secara signifikan. (Barney et al., n.d.)

1.2 Rumusan Masalah

Berdasarkan latar belakang tersebut, maka beberapa rumusan masalah pada tugas akhir ini sebagai berikut :

a. Apakah implementasi komputasi paralel dapat meningkatkan kinerja aplikasi *deep convolutional neural network* (aplikasi *YUV Video Super-Resolution*) ?
b. Bagaimana cara implementasi komputasi paralel pada aplikasi *deep convolutional neural network* (aplikasi *YUV Video Super-Resolution*) ?
c. Bagaimana performa komputasi paralel yang diimplementasikan pada aplikasi *deep convolutional neural network* (aplikasi *YUV Video Super-Resolution*) ?

1.3 Tujuan Penelitian/Perancangan

Tujuan dari penelitian ini adalah berhasil mengimplementasikan komputasi paralel pada aplikasi *YUV Video Super-Resolution* yang diharapkan dapat meningkatkan performa aplikasi/mengurangi waktu yang dibutuhkan untuk memproses video.

1.4 Manfaat Penelitian/Perancangan

Adapun manfaat penelitian yang diharapkan tercapai sebagai berikut :

a. Bagi Penulis

Melalui penulisan tugas akhir ini, peneliti mengembangkan berbagai keterampilan seperti mengubah program sekuensial menjadi program berbasis komputasi paralel, pengumpulan data, pengolahan data, dan analisis data.

b. Bagi Pengguna Aplikasi *YUV Video Super-Resolution*

Melalui implementasi komputasi paralel diharapkan waktu yang dibutuhkan untuk menjalankan aplikasi ini dapat berkurang sehingga dapat menghemat waktu pengguna.

1.5 Batasan Masalah

a. Batasan ruang lingkup :
   - Skripsi ini akan berfokus pada implementasi komputasi paralel pada aplikasi YUV Video Super-Resolution.
   - Implementasi komputasi paralel akan dilakukan dengan menggunakan bahasa pemrograman C.
   - Komputasi paralel akan diimplimentasikan dalam bentuk *multithreading*.
   - Implementasi dan pengambilan data akan dilakukan dengan menggunakan sistem operasi Windows 10 64-bit operating system, x64-based processor.

b. Batasan objek :
   - Skripsi ini akan berfokus pada aplikasi *YUV Video Super-Resolution.*

c. Batasan metodologi :
   - Metode pengumpulan data *CPU time dan wall time* adalah observasi hasil print out bagian *CPU time* dan *wall time* dari aplikasi.
   - Metode pengumpulan data *PSNR* adalah observasi hasil print out bagian *PSNR* dari aplikasi *ffmpeg*.

d. Batasan teknologi :

- Implementasi komputasi paralel akan dilakukan dengan menggunakan perangkat lunak *Visual Studio Code* dan *library OpenMP*.
- Implementasi komputasi paralel akan dilakukan dengan menggunakan *compiler GCC* (*GNU Compiler Collection*) versi 8.2.0.
- Implementasi komputasi paralel dan pengambilan data akan dilakukan pada laptop *Asus* Model *Zenbook* UM431DA dengan spesifikasi prosesor AMD Ryzen 7 3700U (4 *Cores*/8 *Threads*, 6MB *cache*, 4.0GHz) dengan Radeon Vega Mobile Gfx 2.30 GHz dan *random access memory* (RAM) sebesar 8.00 GB tipe DDR4.
- Pengambilan data *PSNR* akan menggunakan aplikasi *open source ffmpeg*.

BAB II
TINJAUAN PUSTAKA

2.1 Komputasi Parallel

Konsep komputasi paralel pertama ditemukan sekitar tahun 1950-an dan mulai banyak digunakan pada tahun 1970-1980-an , lalu pada tahun 1980-an dengan munculnya sistem komputasi dengan performa tinggi, penggunaan komputasi paralel juga meningkat pesat. Komputasi paralel merupakan konsep dimana beberapa komputasi/proses dilakukan secara paralel/bersamaan oleh beberapa unit komputasi, dengan demikian proses yang biasanya dijalankan secara sekuensial dapat langsung berjalan bersamaan sehingga mengurangi waktu yang dibutuhkan untuk mengeksekusi program dan meningkatkan performa aplikasi. Salah satu langkah pertama dalam komputasi paralel adalah memecahkan masalah besar menjadi beberapa bagian masalah kecil yang dapat dikomputasi secara independen (proses komputasinya tidak bergantung satu sama lain) yang dapat dieksekusi secara bersamaan (paralel) oleh beberapa unit komputasi. (Scott et al., 2021)

Sehingga dengan mengimplementasikan komputasi paralel, dapat meningkatkan kecepatan komputasi, efisiensi, dan skalabilitas karena dapat menggunakan beberapa unit komputasi yang tersedia untuk memecahkan suatu masalah. Dibandingkan dengan komputasi serial, komputasi paralel dapat meningkatkan produktivitas dengan memaksimalkan penggunaan sumber daya (unit komputasi) dengan cara mengalokasikan tugas untuk masing-masing unit komputasi. (Scott et al., 2021)

Beberapa tantangan yang dihadapi dalam implementasi komputasi paralel:

1. Limitasi perangkat keras dan lunak : komputasi paralel untuk masalah atau dataset yang besar membutuhkan perangkat keras yang sesuai, selain itu perangkat lunak dari perangkat tersebut juga harus efisien dalam memberikan tugas ke masing-masing unit komputasi (Barney et al., n.d.)
2. Pemrograman yang lebih rumit : untuk mengubah program dengan komputasi konvensional dibutuhkan pemahaman yang lebih dalam mengenai pembagian data, bagaimana perhitungan setiap iterasi dilakukan, dan

menentukan klasifikasi variabel data (deklarasi variabel data *private* atau *shared*) dari variabel-variabel bagian program yang akan diparalelkan, oleh karena itu untuk mengimplementasi komputasi paralel pada suatu program dengan komputasi konvensional lebih rumit. (Intel & Gillespie, n.d.)

3. Tidak mudah mendapatkan *performance gain* atau *speedup* yang mendekati linear : Disebabkan oleh pada umumnya hanya beberapa bagian kode sebuah program yang dapat dieksekusi secara paralel sehingga sebagian besar kode hanya dapat dijalankan secara sekuensial selanjutnya ada faktor-faktor lain yang disebut dengan parallelization overhead yang meliputi pembuatan thread, penghilangan thread, sinkronisasi thread, dan penguncian data tertentu agar data tersebut tidak diubah oleh banyak thread diwaktu yang bersamaan. (Intel & Gillespie, n.d.)

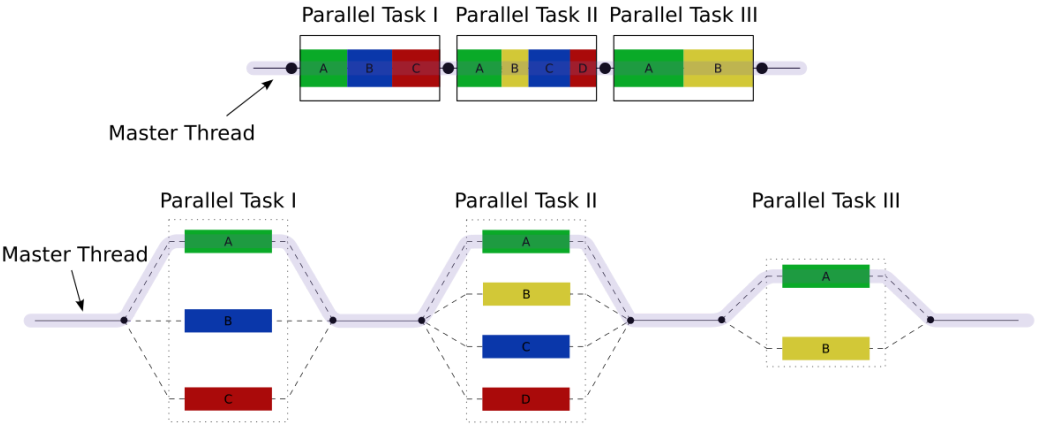
2.2 *OpenMP*

OpenMP merupakan salah satu *API* (*Application Program Interface*) yang dapat digunakan untuk mengembangkan aplikasi berbasis komputasi paralel yang mengimplementasikan *multithreading* dalam bahasa pemrograman *C*, *C++*, dan *Fortran*. *Multithreading* merupakan metode untuk memparalelkan sebuah program dimana sebuah *thread* utama membuat beberapa sub-*thread* dan sistem kemudian akan membagikan tugas pada *thread-thread* tersebut. Lalu setiap *thread* akan menjalankan tugasnya masing-masing secara bersamaan/konkuren. (OpenMP Architecture Review Board, 2023)

OpenMP diterbitkan pada bulan Oktober tahun 1997 oleh *OpenMP Architecture Review Board (ARB)* yang pada awal hanya tersedia dalam bahasa pemrograman Fortran. Satu tahun setelah itu *OpenMP ARB* kemudian merilis juga API dalam bahasa pemrograman C/C++. Versi 2.0 untuk Fortran diterbitkan pada tahun 2000 dan untuk C/C++ pada tahun 2002. Lalu pada tahun 2005 versi 2.5 dirilis spesifikasi untuk gabungan Fortran dan C/C++. Untuk versi selanjutnya 3.0, dirilis pada bulan Mei 2008. (*OpenMP Architecture Review Board*, 2023)

Dalam *OpenMP*, untuk menandakan bagian kode yang harus dijalankan secara paralel dapat digunakan sebuah *compiler directive* dengan tujuan membentuk beberapa *thread* sebelum bagian kode yang telah ditandai dengan

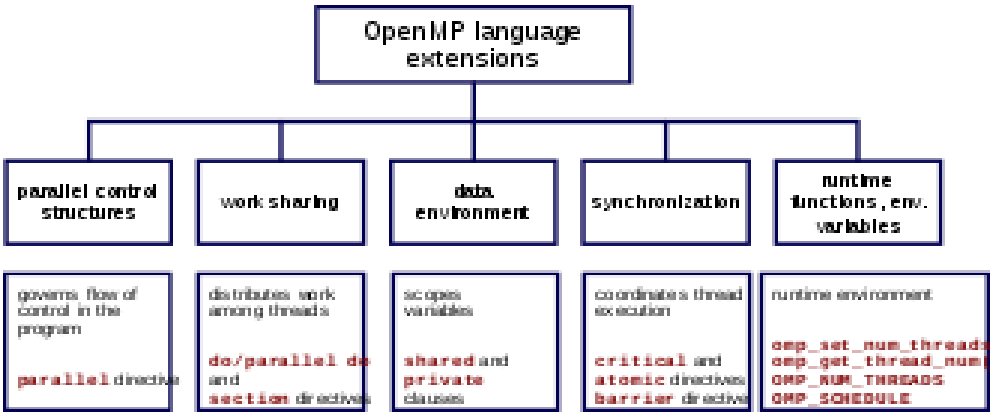
compiler directive dieksekusi ditambahkan dengan klausa untuk memperjelas pembagian tugas. Setiap sub-*thread* memiliki ID yang berbeda (1,2,3, dan seterusnya) dan untuk *thread* utama (*master thread*) memiliki ID 0. Setelah eksekusi bagian kode paralel telah selesai, seluruh *subthread* kembali bergabung dengan *thread* utama (Gambar 2.1). (OpenMP Architecture Review Board, 2023)



Gambar 2.1 Ilustrasi pembagian tugas paralel

2.2.1 *Directives* (Direktif)

Directives (Direktif) dalam *OpenMP* digunakan untuk menandakan bagian kode yang harus dijalankan secara paralel. Direktif dimulai dengan sintaks : *#pragma omp* , dan dilanjutkan dengan pembuatan *thread*, *work-sharing construct*, dan klausa-klausa yang relevan (Gambar 2.2). (OpenMP Architecture Review Board, 2023)



Gambar 2.2 Bagan konstruk *OpenMP*

Untuk pembuatan *thread* ditambahkan *parallel* pada sintaks direktif menjadi : *#pragma omp parallel* , dengan tujuan untuk membuat *thread* tambahan untuk mengeksekusi bagian kode yang paralel. Selanjutnya untuk melengkapi sintaks direktif akan ditambahkan sintaks untuk proses pembagian tugas masing-masing *thread* yang disebut dengan *work-sharing construct*. *Work-sharing construct* terbagi atas 4, yaitu :

1. *"omp for"* atau "*omp do"*

   Digunakan untuk membagi iterasi-iterasi dari perulangan *for* atau *do* antar *thread*.

   Sintaks : *#pragma omp parallel for* atau *#pragma omp parallel do*

2. *"sections"*

   Membagi tugas berbeda kepada beberapa *thread* untuk mengeksekusi bagian kode paralel yang independen satu sama lain namun berurutan.

   Sintaks : *#pragma omp parallel sections*

3. *"single"*

   Menandakan bagian kode yang ditandai dengan *construct* ini hanya boleh dijalankan/dieksekusi oleh satu *thread*. Pada umumnya diikuti oleh direktif klausa *barrier* pada akhir bagian kode tersebut.

   Sintaks : *#pragma omp single*

4. *"master"*

   Mirip dengan *construct 'single'*, namun yang membedakan *construct 'master'* dari '*single'* adalah bagian kode yang ditandai dengan *construct 'master'* dijalankan/dieksekusi oleh *thread master* (utama).

   Sintaks : *#pragma omp master*

(OpenMP Architecture Review Board, 2023)

2.2.2 *Clause* (Klausa)

Dalam *OpenMP*, *clause* (klausa) merujuk pada aturan dan atribut yang digunakan untuk mengatur bagaimana variabel dalam kode *OpenMP* berperilaku terhadap berbagai *thread*. Dalam konteks *OpenMP*,

sebagian besar variabel secara *default* terlihat oleh semua *thread* (*shared*), tetapi dalam situasi tertentu, akan ada variabel yang harus hanya terlihat oleh satu *thread* untuk menghindari terjadinya konflik, atau ketika perlu memindahkan nilai sebuah variabel dari bagian sekuensial ke wilayah paralel (bagian kode yang dijalankan secara paralel). (*OpenMP Architecture Review Board*, 2023) Untuk mengatasi masalah tersebut, dapat digunakan klausa atribut pembagian data dengan menambahkannya pada direktif *OpenMP*. Terdapat beberapa jenis klausa yang dapat digunakan sebagai berikut (*OpenMP Architecture Review Board*, 2023):

1. *Data sharing attribute clauses* (Klausa atribut pembagian data)
   - `*shared*`: Klausa ini mengindikasikan bahwa data yang dideklarasikan di luar wilayah paralel bersifat *shared*, yang berarti variabel ini dapat diakses oleh semua *thread* secara bersamaan. Secara default, hampir semua variabel dalam wilayah berbagi kerja bersifat *shared*, kecuali untuk variabel penghitung perulangan *loop*.
   - `*private*`: Klausa ini digunakan agar setiap *thread* akan memiliki salinan lokal dan menggunakannya sebagai variabel sementara. Variabel penghitung perulangan dalam konstruksi loop *OpenMP* secara default bersifat *private*.
   - `*default*`: Klausa ini memungkinkan pembuat program untuk menentukan cakupan data *default* dalam wilayah paralel, apakah itu bersifat *shared* atau *none* dalam konteks pemrograman dengan menggunakan bahasa C/C++ dan *shared, firstprivate, private,* or *none* untuk Fortran. Opsi "*none*" memaksa pembuat program untuk mendeklarasikan setiap variabel dalam wilayah paralel menggunakan klausa atribut berbagi data.
   - `*firstprivate*`: klausa ini mirip dengan klausa "*private*," tetapi variabel diinisialisasi menggunakan nilai asli.
   - `*lastprivate*`: Sama seperti "*private*," namun nilai asli diperbarui setelah *construct*.

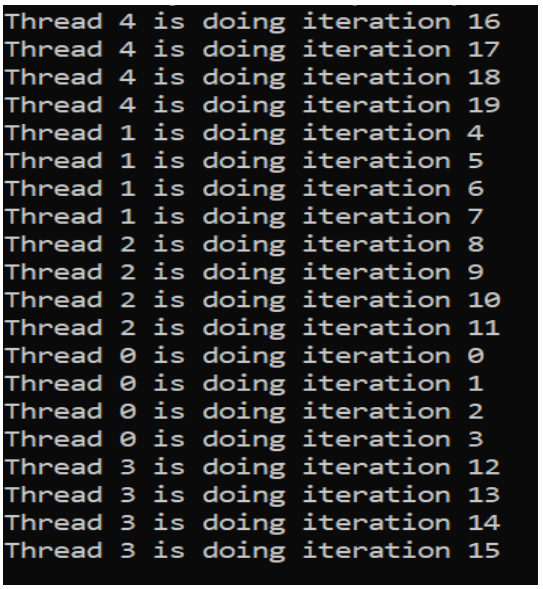
- `*reduction*`: Klausa ini adalah klausa *data-sharing* yang dapat digunakan untuk melakukan kalkulasi dalam bentuk rekuren secara paralel. Klausa ini digunakan untuk menggabungkan hasil kalkulasi masing-masing *thread* dari sebuah variabel yang bersifat *shared* sesuai dengan operator yang digunakan, misalkan operator yang digunakan adalah tanda +, maka klausa *reduction* pada akhir wilayah paralel akan menjumlahkan seluruh hasil kalkulasi setiap *thread*. Dengan demikian klausa *reduction* dapat digunakan untuk memastikan tidak terjadinya situasi dimana beberapa *thread* disaat yang bersamaan memodifikasi sebuah variabel *shared,* situasi ini dikenal dengan istilah *race condition*.

2. *Synchronization clauses* (Klausa sinkronisasi)
   - `*critical*`: Klausa ini memastikan bahwa blok kode akan dieksekusi oleh hanya satu *thread* pada satu waktu untuk melindungi data yang bersifat *shared* dari *race condition* (sebuah kondisi dimana beberapa *thread* mengakses sebuah data variabel yang sama dan ingin mengubah nilai data secara bersamaan).
   - `*atomic*`: Klausa '*atomic*' memastikan bahwa pembaruan memori dalam instruksi selanjutnya akan dilakukan secara atomik, namun tidak menjadikan keseluruhan pernyataan menjadi atomik. Dengan demikian biasanya memiliki performa yang lebih baik dibandingkan klausa *critical.*
   - `*ordered*`: Klausa '*ordered*' memastikan bahwa blok terstruktur dieksekusi dalam urutan yang sesuai dengan urutan iterasi dalam loop sekuensial.
   - `*barrier*`: Klausa `*barrier*` membuat setiap *thread* menunggu hingga semua *thread* lain mencapai titik tertentu, biasanya digunakan untuk mengakhiri *work-sharing construct*.
   - `*nowait*`: Digunakan untuk menginstruksikan *thread* agar melanjutkan tugas tanpa harus menunggu *thread* lainnya menyelesaikan tugasnya masing-masing.

3. *Scheduling clauses* (Klausa penjadwalan)

   Digunakan untuk *work-sharing construct* dengan bentuk perulangan for loop atau do loop. Ada 3 jenis klausa penjadwalan :

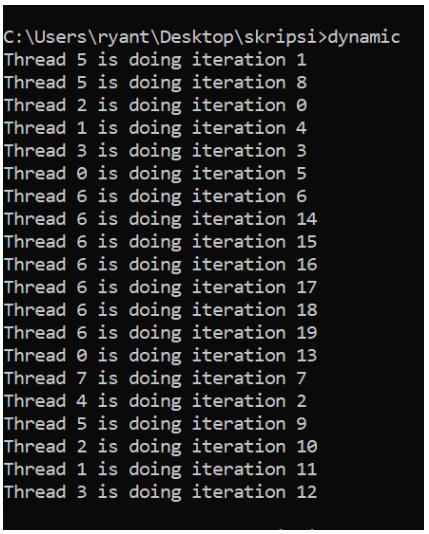
   - ` *static* ` : untuk penjadwalan jenis *static*, semua *thread* dibagikan iterasi *loop,* masing-masing dengan jumlah yang sama rata secara *default*. Jika ingin menspesifikasikan berapa banyak jumlah iterasi yang ingin berikan pada masing-masing *thread* juga dapat dilakukan dengan mengatur nilai parameter *chunk* dalam klausa static, sebagai contoh : *schedule*(jenis_*schedule*, *chunk*) ~> *schedule*(*static*, 4) , maka masing-masing *thread* akan mengerjakan masing-masing 4 iterasi (*thread* 0 akan mengerjakan iterasi 0,1,2,3 dan *thread* 1 akan mengerjakan *thread* 4,5,6,7 dan seterusnya, dapat dilihat pada gambar 2.3)



Gambar 2.3 Hasil eksekusi *static scheduling*

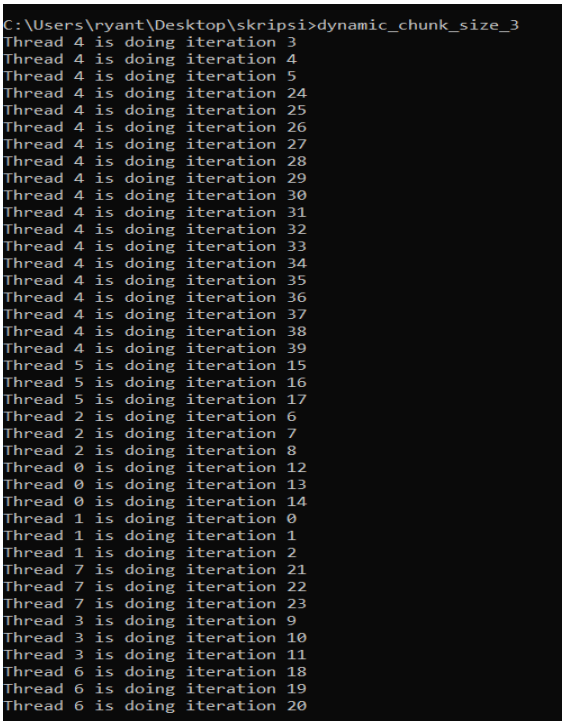
   - ` *dynamic* ` : pembagian tugas sesuai berdasarkan ketersediaan *thread*, dengan kata lain *thread-thread* yang sedang tidak menjalankan tugas (yang tersedia) akan diberikan tugas, dengan demikian memungkinkan pembagian tugas tidak sama rata karena apabila sebuah *thread* yang sudah pernah mendapat tugas telah menyelesaikan tugasnya maka memungkinan *thread* tersebut akan

diberikan tugas lagi, untuk contoh dynamic scheduling tanpa *chunk* size dapat dilihat pada gambar 2.4



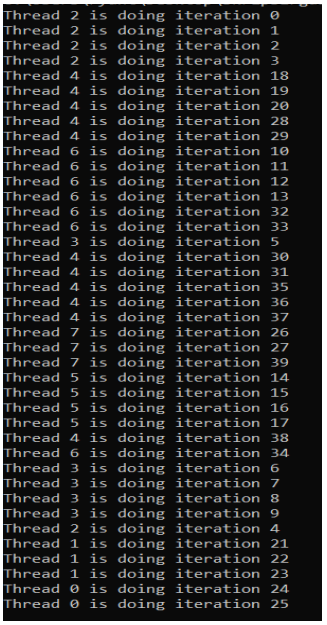
Gambar 2.4 Hasil eksekusi *dynamic scheduling*

Untuk *dynamic scheduling* dengan *chunk size n*, beberapa *thread* akan mendapatkan tugas sebanyak iterasi *n*, dan jika *thread* sudah melakukan tugasnya, memungkinan *thread* tersebut mendapatkan tugas lagi sebesar *n* atau lebih kecil dari *n* jika jumlah iterasi *modulo n* memiliki sisa bagi, untuk ilustrasinya dapat dilihat pada gambar 2.5 yang memiliki dynamic scheduling dengan *chunk size* 3.



Gambar 2.5 Hasil eksekusi *dynamic scheduling* dengan *chunk size* 3

- ` *guided* ` : menyerupai *dynamic scheduling* dalam aspek pemberian tugas kepada *thread* yang sedang tidak memiliki tugas, ketika sebuah *thread* telah menyelesaikan tugasnya, memungkinan *thread* tersebut diberikan tugas lagi, namun ukuran tugas awal yang diberikan berukuran besar akan semakin kecil apabila diberikan tugas lagi (Gambar 2.6)



Gambar 2.6 Hasil eksekusi *guided scheduling*

4. *IF control* (Klausa pengendalian "*if*")
   - ` *if* `: Klausa ini berlaku jika kondisi yang telah ditentukan terpenuhi, kode akan dieksekusi/dijalankan secara paralel, dan jika kondisi tidak terpenuhi maka blok kode akan dijalankan secara sekuensial
5. *Initialization* (Inisialisasi)
   - `*firstprivate*`: mirip dengan atribut data "*private*", namun digunakan untuk inisialisasi nilai variabel sebelum bagian kode paralel atau dengan kata lain nilai yang disalin pada masing-masing *thread* merupakan nilai awal yang sama dari master *thread*.
   - `*lastprivate*`: menyerupai klausa "*firstprivate*," tetapi nilai data "*private*" akan disalin ke variabel global untuk iterasi terakhir

dalam loop yang diparalelkan, sehingga nilai yang disalin ke variabel global merupakan nilai akhir dari iterasi terakhir.

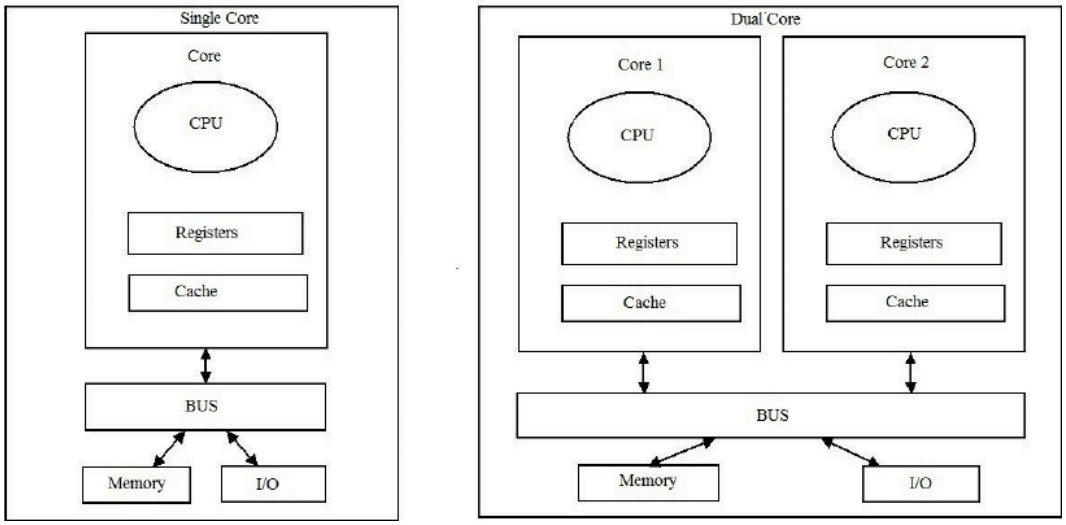
- `*threadprivate*`: Data variabel bersifat global, tetapi menjadi "*private"* dalam semua wilayah paralel selama *runtime*. Perbedaan utama dengan atribut data "*private*" adalah ruang lingkup global yang terkait dengan "*threadprivate*" dan nilainya dipertahankan di seluruh wilayah paralel.

6. *Data copying* (Penyalinan data)
   - `*copyin*` : menyalin nilai variabel dengan atribut "*threadprivate*" dari *thread* utama ke semua *thread*.
   - `*copyprivate*` : digunakan dengan construct single untuk menyalin data dari *thread* yang menjalankan construct single tersebut ke *thread* lainnya.
7. *Reduction* (Reduksi)
   - `*reduction*` : Klausa ini digunakan untuk menggabungkan hasil kalkulasi dari masing-masing *thread* sesuai dengan operator yang digunakan, misalkan operator yang digunakan adalah tanda +, maka klausa *reduction* pada akhir wilayah paralel akan menjumlahkan seluruh hasil kalkulasi setiap *thread*.
8. Klausa lainnya
   - `*flush*` : digunakan untuk memastikan semua *thread* yang bekerja pada variabel *shared* menggunakan nilai yang terkini dan konsisten pada semua *thread*.
   - `*master*` : bagian kode yang ditandai dengan *construct 'master'* hanya boleh dijalankan/dieksekusi oleh *thread master* (utama).
   - `*simd*`: : digunakan untuk menandakan bahwa beberapa iterasi loop (perulangan) tersebut dapat dijalankan disaat yang bersamaan/konkuren

- `*collapse*` : klausa ini digunakan untuk mengubah nested loop (loop yang ada dalam loop lainnya) menjadi satu loop (menyatukan loop menjadi satu) dengan tujuan untuk paralelisasi

(*OpenMP Architecture Review Board*, 2023)

2.3 *Multicore Processor*



Gambar 2.7 Diagram Blok *single core processor* dan *dual core processor* (*multicore processor*)

(Saini, 2015)

Multicore processor merupakan sebuah jenis prosesor terdiri atas beberapa unit komputasi (core) yang dapat bekerja secara independen disaat yang bersamaan, sehingga memungkinkan terjadinya proses eksekusi tugas secara paralel. Salah satu keuntungan penggunaan *multicore processor* dibandingkan dengan prosesor biasa (*single core processor*) yaitu membutuhkan daya listrik yang lebih kecil dan waktu yang lebih singkat dibandingkan prosesor biasa untuk mengerjakan tugas yang sama, dengan demikian *multicore processor* memiliki performa yang jauh lebih baik dibandingkan *single core processor* oleh karena terdapat beberapa unit komputasi (core) yang dapat mengeksekusi beberapa instruksi/tugas disaat yang bersamaan. (Anil Sethi, 2015) Beberapa tantangan yang muncul dari penggunaan multicore processor adalah :

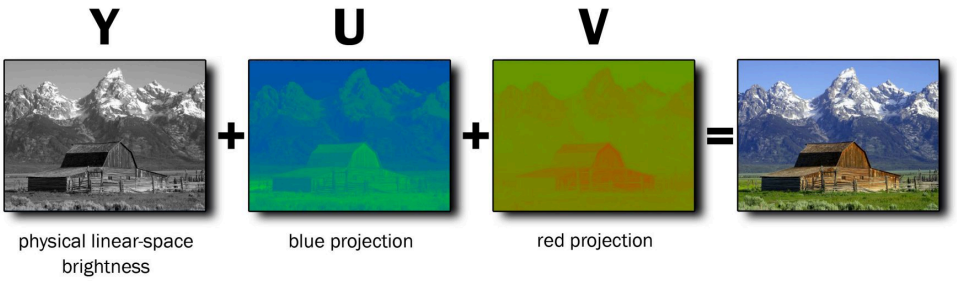
1. Penggunaan daya dan temperatur : daya listrik yang dikonsumsi oleh sistem berbanding lurus dengan temperatur yang dihasilkan oleh sistem, seiring meningkatnya konsumsi daya maka temperatur sistem tersebut juga akan meningkat hingga pada titik *overheat* (sistem menjadi terlalu panas). Selain itu, oleh karena prosesor jenis *multicore* terdiri atas beberapa unit komputasi (*core*) dalam satu die (tempat sirkuit prosesor yang terbuat dari bahan silikon), maka prosesor jenis *multicore* akan memakan daya yang lebih tinggi dibandingkan *single core processor* jika tidak diberlakukan skema optimasi penggunaan daya seperti menjalankan *core* dalam mode frekuensi yang lebih rendah atau memutuskan daya *core* yang sudah digunakan. (Shruti Jadon, 2016)
2. Utilisasi *multicore* : utilisasi *multicore* merujuk pada utilisasi keseluruhan semua *core* yang tersedia, sehingga nilai utilisasi total tidak lebih dari 1. Tantangan yang datang dari utilisasi *multicore* adalah mencapai utilisasi yang hampir sama atau sama untuk semua *core* yang tersedia serta disaat yang bersamaan menggunakan masing-masing *core* secara maksimal, dengan kata lain beban kerja yang diberikan harus terdistribusi secara merata untuk seluruh *core* dan tidak hanya membebankan 1 *core* (*load balancing*). Untuk mengatasi masalah *load balancing* dapat diimplementasikan algoritma *load balancing* untuk memastikan seluruh core mendapatkan beban kerja yang sama. (Shruti Jadon, 2016)

2.4 Aplikasi *YUV Video Super-Resolution*

Aplikasi *YUV Video Super-Resolution* merupakan aplikasi yang digunakan untuk meningkatkan resolusi video dengan format YUV sebesar 2 kali lipat. Aplikasi *YUV Video Super-Resolution* menggunakan algoritma berbasis *Deep Convolutional Neural Network* (*DCNN*) yaitu *Fast Super Resolution Convolutional Neural Network* (*FSRCNN*) dalam mengolah video. Untuk neural network yang digunakan dalam aplikasi ini sudah dilatih dan bisa langsung digunakan dengan memasukkan *input* video dengan format YUV dan akan menghasilkan *output* video dengan format YUV yang memiliki resolusi 2 kali lebih besar dari video *input*. (Abdollahzadeh, 2022)

2.4.1 Format Video YUV

Istilah YUV mengacu pada rangkaian ruang warna, dimana YUV sesuai dengan namanya terdiri atas 3 komponen, 1 komponen tingkat kecerahan warna (luma) yaitu komponen Y, dan 2 komponen warna (chroma) U (chroma blue/ biru) dan V (chroma red/ merah) (Gambar 2.8). Setiap *pixel* dalam video YUV direpresentasikan oleh 3 komponen tersebut dalam bentuk data biner yang disusun dengan urutan tertentu. (DEXON Systems, 2022)



Gambar 2.8 Ilustrasi komponen penyusun video format YUV

Aplikasi *YUV Video Super-Resolution* menggunakan video YUV dengan format YUV420 8-bit dimana angka 4 merujuk pada informasi yang disimpan untuk tingkat kecerahan (*luma*) atau komponen Y tetap dipertahankan, namun untuk komponen chroma (*warna*) memiliki setengah (2) dari resolusi komponen Y. Tujuan penggunaan format tersebut adalah untuk mengurangi jumlah data yang harus disimpan untuk setiap *pixel*, walaupun demikian gambar yang dihasilkan tidak mengalami penurunan kualitas jika diamati oleh mata manusia, dikarenakan mata manusia lebih sensitif terhadap perubahan tingkat kecerahan dibandingkan perubahan warna. (Microsoft, 2022)

2.4.2 Cara Kerja Aplikasi / Alur Kerja Aplikasi

Aplikasi YUV Video Super Resolution menggunakan algoritma FSRCNN sehingga terdiri atas 5 bagian utama yaitu :

1. Ekstraksi fitur dari gambar resolusi rendah (*LR*) ~ *Feature extraction* : *Conv(5,56,1)*

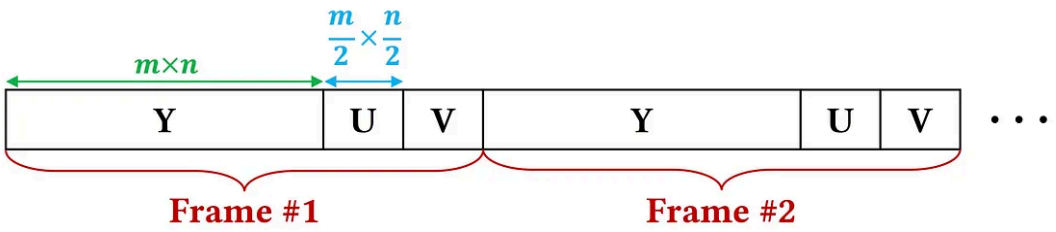
2. Selanjutnya dimensi fitur LR akan diperkecil ~ *Shrinking* : *Conv(1,12,56)*
3. Lalu dilanjutkan dengan proses pemetaan dari dimensi fitur LR ke HR, proses pemetaan ini terdiri atas 4 *layer* konvolusi ~ *Non-linear mapping* : *4 × Conv(3,12,12)*
4. Setelah itu dimensi fitur-fitur HR ditingkatkan agar dapat digunakan untuk rekonstruksi gambar ~ *Expanding* : *Conv(1,56,12)*
5. Akhirnya dengan proses dekonvolusi, gambar meningkatkan resolusi fitur-fitur HR dan menggabungkannya dengan menggunakan filter dekonvolusi ~ *DeConvolution* : *DeConv(9,1,12)*

(Abdollahzadeh, 2022)

Selanjutnya, setiap proses/bagian utama pada aplikasi ini disertai dengan fungsi aktivasi. Fungsi aktivasi yang digunakan adalah *PReLU*. Sehingga pada aplikasi ini, *neural network* yang digunakan terdiri atas 8 *layer* yang dinotasikan sebagai berikut (Abdollahzadeh, 2022):

Conv(5,56,1) — PReLU — Conv(1,12,56) — PReLU — 4 × {Conv(3,12,12) — PReLU — Conv(1,56,12) — PReLU — DeConv(9,1,12)

Sebelum aplikasi menjalankan algoritma *FSRCNN*, aplikasi akan membaca video *input* sebagai file biner dengan memanfaatkan fungsi *fread* dari *library* bahasa pemrograman C, dimana untuk tiap *frame* terdapat data untuk komponen Y, U, dan V secara berurutan. Untuk video dengan resolusi spasial m x n dapat dilihat pada gambar 2.9. (Abdollahzadeh, 2022)

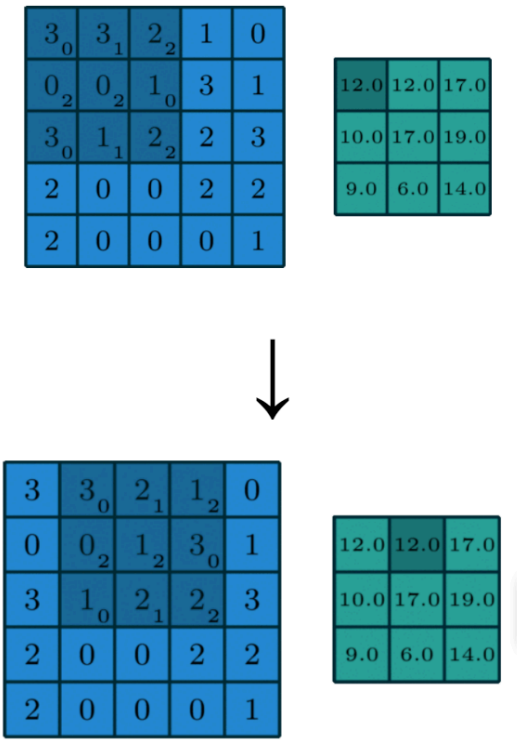


Gambar 2.9 Video YUV sebagai *file* biner

Lalu untuk menjalankan algoritma *FSRCNN*, terdapat beberapa operasi (fungsi) dalam aplikasi yang dapat dirincikan sebagai berikut :

1. *Convolution*

Dalam proses konvolusi terdapat matriks *weight* yang dikenal dengan sebutan kernel. Kernel tersebut digunakan untuk perkalian matriks dengan bagian kecil dari *input* gambar, proses ini dilakukan berulang kali dengan cara menggeser kernel sesuai dengan parameter stride (k) (parameter dalam *FSRCNN* yang menentukan seberapa besar penggesaran filter terhadap gambar, misalkan $k = 1$ maka *filter* akan bergeser sebesar 1 *pixel* ke bagian gambar selanjutnya) hingga seluruh bagian gambar terkena perkalian matriks dengan kernel tersebut. (Gambar 2.10)

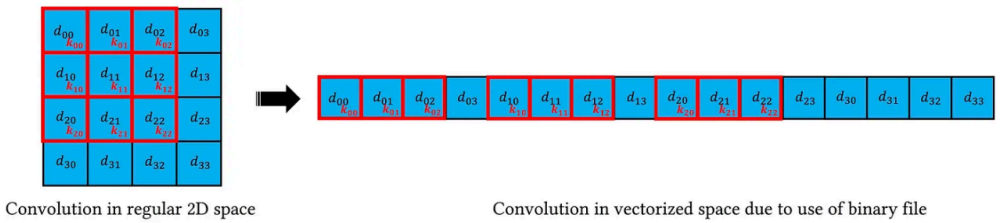


Gambar 2.10 Ilustrasi proses konvolusi

Persegi warna biru adalah representasi gambar dalam bentuk matriks dan angka kecil bagian kanan bawah masing-masing persegi kecil (*pixel*) merupakan filter/kernel, persegi warna hijau merupakan hasil perkalian matriks dari gambar *input* dengan kernel.

Oleh karena data *input* merupakan file biner, maka perkalian matriks kernel dengan data *input* tidak dapat langsung dilakukan, melainkan proses konvolusi akan dilakukan dalam ruang vektor. Gambar 2.11

mengilustrasikan bagaimana pengaplikasian kernel konvolusi pada file biner dilakukan untuk data *input* 4 x 4 dengan kernel berukuran 3 x 3 :



Gambar 2.11 Proses konvolusi dalam ruang vektor

2. Fungsi aktivasi *PReLU (Parametric Rectified Linear Unit)*

   Fungsi aktivasi digunakan untuk menentukan aktivasi sebuah *neuron*/*node*. Fungsi aktivasi yang pada umumnya digunakan yaitu *ReLU* (*Rectified Linear Unit*). Namun *ReLU* memiliki masalah dimana ada sekumpulan node/neuron yang stagnan dikarenakan *ReLU* akan secara otomatis menetapkan *input* bernilai negatif menjadi 0, dengan kata lain *node/neuron* tersebut tidak mengalami aktivasi dan memungkinan nilai *weight neuron* tersebut tidak berubah selama proses pembelajaran. *PReLU (Parametric Rectified Linear Unit)* dapat digunakan untuk menghindari masalah ini, dikarenakan *PReLU* tidak mengabaikan *input* yang bernilai negatif, sehingga menghindari adanya *neuron* yang mati/stagnan. *PReLU* dapat didefinisikan sebagai berikut :

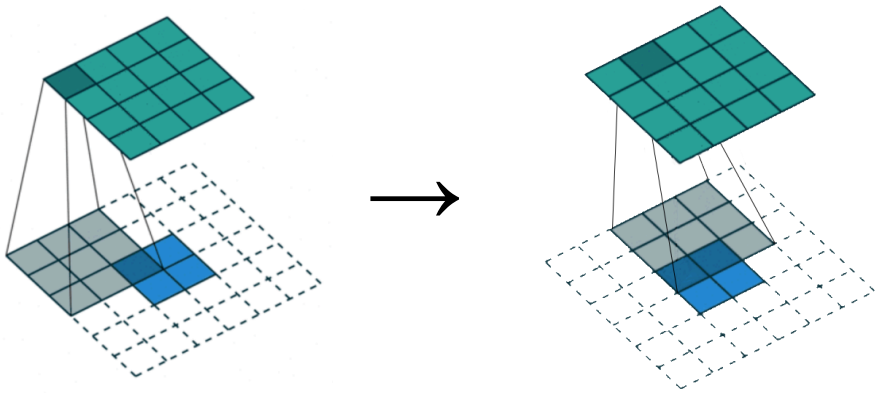
   $f(x_i) = max(x_i, 0) + a_i min(x_i, 0)$

   Parameter a_i merupakan nilai *weight* dari *input* bernilai negatif dan merupakan parameter yang dapat dipelajari oleh jaringan.

3. *Deconvolution*

   Proses dekonvolusi merupakan invers dari konvolusi. Pada operasi konvolusi, dengan menggunakan parameter "*stride* (k)" (parameter dalam yang menentukan seberapa besar penggesaran filter terhadap gambar, misalkan k = 1 maka filter akan bergeser sebesar 1 *pixel* ke bagian gambar selanjutnya), filter dikonvolusi dengan gambar sesuai nilai k, maka akan menghasilkan *output 1/k* dari *input*. Oleh karena operasi dekonvolusi

merupakan kebalikan dari konvolusi, posisi *input* dan *output* dari operasi konvolusi dibalik sehingga *output* menjadi *k* kali *input*. Berdasarkan *output* tersebut, jika nilai *k* ditentukan nilainya yaitu *k* = *n*, dimana *n* merupakan faktor upscaling (seberapa besar peningkatan kualitas gambar), dengan demikian *output* yang langsung didapatkan adalah gambar resolusi tinggi (*HR*) yang direkonstruksi sesuai dengan faktor *upscaling* n (misalkan faktor *upscaling n* bernilai dua maka gambar yang direkonstruksi akan menjadi dua kali lebih besar atau resolusinya akan meningkat 2 kali lipat).



Gambar 2.12 Ilustrasi proses dekonvolusi

Gambar 2.12 mengilustrasikan bagaimana proses dekonvolusi terjadi, persegi warna hijau merupakan fitur-fitur yang didapatkan pada *layer* sebelumnya kemudian digabung menggunakan filter dekonvolusi untuk mendapatkan gambar HR (persegi dengan gari putus).

(Abdollahzadeh, 2022)

2.5 *Artificial Neural Network (ANN)*

Artificial Neural Network (ANN) merupakan salah satu tipe machine learning yang dibuat berdasarkan inspirasi dari jaringan saraf biologis pada manusia dan hewan. Dalam *ANN* terdapat kumpulan elemen pemrosesan yang dikenali dengan sebutan *neuron* atau *node*, dan koneksi antar node terdapat koefisien yang disebut *weight* dan nilai konstan disebut *bias* yang terikat kepada koneksi-koneksi tersebut. (Choi et al, 2020)

Gagasan awal tentang jaringan saraf berakar pada tahun 1873, ketika Alexander Bain pertama kali mengemukakan teorinya berdasarkan kemajuan dalam studi neuroanatomi. Bain berpendapat bahwa otak manusia terdiri dari neuron-neuron yang saling terhubung, dan setiap aktivitas otak melibatkan pola tertentu dari neuron-neuron tersebut beserta koneksi antar-neuron yang terkait. Bain juga menyadari bahwa pengulangan pikiran, sensasi, atau pengalaman tertentu cenderung memperkuat koneksi tersebut, sebuah gagasan yang kemudian dikenal sebagai "aturan asosiasi." Pemikiran ini menjadi dasar banyak teori neural modern yang kita gunakan hingga saat ini. (Pirez et al, 2023)

Pada tahun 1943, Warren McCulloch dan Walter Pitts menciptakan model jaringan saraf tiruan pertama yang terinspirasi oleh cara kerja neuron di otak. Mereka memanfaatkan konsep seperti aktivasi neuron dan potensial aksi biner untuk membangun struktur sederhana yang mampu melakukan operasi logika biner. Dengan menggabungkan unit-unit dasar ini, mereka berhasil menunjukkan bahwa berbagai kombinasi logika biner dapat disimulasikan menggunakan jaringan. (Pirez et al, 2023)

Kemajuan besar berikutnya datang dari Donald Hebb pada tahun 1949. Ia mengembangkan teori bahwa ketika satu *neuron* terus-menerus mengaktifkan *neuron* lain, koneksi di antara keduanya akan semakin kuat—sebuah prinsip yang kini dikenal sebagai aturan *Hebb*. Ide ini menjadi pondasi untuk algoritma komputer yang mulai muncul pada awal 1950-an, membuka jalan bagi pengembangan lebih lanjut di bidang kecerdasan buatan. (Pirez et al, 2023)

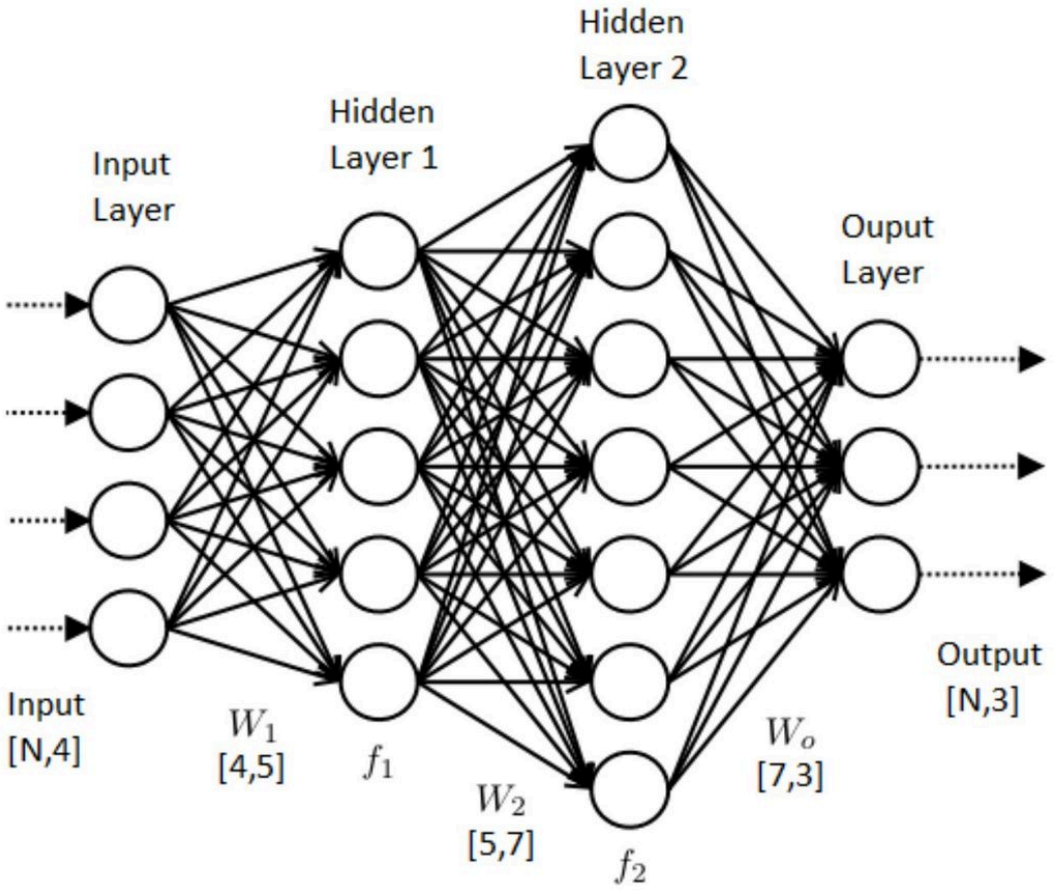
Pada dekade 1960-an, Frank Rosenblatt memperkenalkan sebuah jenis jaringan saraf baru yang disebut Perceptron. Teknologi ini memungkinkan jaringan untuk "belajar" dengan mencocokkan input tertentu dengan output yang

sesuai melalui proses pelatihan. Rosenblatt bahkan membuktikan bahwa jika sebuah masalah memiliki solusi, *Perceptron* dapat menemukannya dalam jumlah iterasi yang terbatas. Namun, di akhir 1960-an, beberapa batasan *Perceptron* ditemukan, terutama dalam menyelesaikan masalah kompleks yang memerlukan pendekatan lebih canggih. (Pirez et al, 2023)

Pada periode yang sama, Bernard Widrow dan Marcian Hoff mengembangkan metode pembelajaran yang disebut aturan *Widrow-Hoff* atau aturan *Delta*. Metode ini memperluas kemampuan jaringan dengan memungkinkan pengolahan data yang tidak hanya berasal dari data pelatihan. Awalnya, metode ini hanya diterapkan pada jaringan satu lapisan, tetapi seiring waktu, arsitekturnya diperluas untuk mendukung jaringan multi-lapisan yang lebih kompleks. (Pirez et al, 2023)

Selama evolusi jaringan saraf tiruan, jelas bahwa konsep jaringan saraf tiruan selalu berakar pada inspirasi dari biologi. Hubungan mendalam antara cara kerja otak manusia dan model komputasi modern terus menjadi inti dari perkembangan teknologi kecerdasan buatan hingga hari ini. (Pirez et al, 2023)

ANN terdiri atas sebuah *layer node input*, satu *layer node output*, dan diantara kedua *layer* tersebut terdapat satu atau beberapa *layer* node tersembunyi (*hidden layers*), dimana setiap masukan yang masuk ke *layer input* akan dikalkulasikan berdasarkan *weight* dan *bias* yang terdapat dalam *layer* tersembunyi (*hidden layer*) tersebut sebelum diteruskan ke node / *layer node* berikutnya dan *input* dari *layer* sebelumnya lalu dikalkulasikan berdasarkan *weight* dan *bias* dan seterusnya hingga *hidden layer* terakhir, kemudian diteruskan ke *node* pada *layer output.* (Gambar 2.13). Dengan demikian *ANN* dapat belajar melalui penyesuain *weight* dan *bias* dengan membandingkan hasil prediksi dan hasil yang diharapkan berdasarkan data-data yang ada. Metode pembelajaran ANN tersebut disebut dengan *supervised learning.* (Choi et al, 2020)



Gambar 2.13 Artificial Neural Network

Selanjutnya, metode pembelajaran ANN selain *supervised learning*, ada *unsupervised learning* dimana ANN akan mempelajari *input* dengan tujuan mencari hubungan antar variabel data yang ada untuk membuat sebuah fungsi yang dapat menjelaskan/menggambarkan pola hubungan antar variabel. (Choi et al, 2020)

2.6 *Convolutional Neural Network (CNN)* dan *Deep Convolutional Neural Network (DCNN)*

CNN (Convolutional Neural Network) merupakan salah satu tipe *artificial neural network (ANN)*, dimana *convolutional neural network* berbeda dari *artificial neural network (ANN)* pada umumnya karena dapat menyimpan hubungan antar *pixel* dari sebuah gambar dengan cara menyimpan konteks spasial yang bertujuan untuk mengekstraksi fitur spesifik dari gambar tersebut. Perbedaan *DCNN* dengan *CNN* hanya terletak pada jumlah *layer* konvolusional, dimana

DCNN memiliki jumlah *layer* konvolusional yang lebih banyak (dapat mencapai puluhan hingga ribuan) dibandingkan *CNN*. (Alzubaidi et al, 2021)

Berbeda dengan jaringan saraf tiruan (*Artificial Neural Network/ANN*) yang pada umumnya yang menggunakan sebuah *pixel* sebagai *input*, *CNN* menggunakan potongan gambar (beberapa *pixel* sekaligus) pada node spesifik (tidak semua node) pada *layer* node selanjutnya, kumpulan node tersebut dikenali dengan sebutan *convolutional filters*. Lalu *filter* tersebut digunakan untuk perkalian matriks dengan bagian gambar *input* di berbagai posisi untuk menghasilkan peta fitur baru. Selanjutnya, *Deep Convolutional Neural Network (DCNN)* akan mendapatkan peta fitur baru dengan cara menggunakan fitur yang diekstraksi sebagai *input* selanjutnya dan proses ini dapat terus berulang untuk beberapa *layer* dan semakin banyak perulangan yang terjadi, maka fitur yang diekstrak akan semakin abstrak, hingga akhirnya fitur tersebut dapat digunakan untuk melakukan prediksi. Berdasarkan cara kerja *DCNN* tersebut maka, *DCNN* seringkali digunakan dalam bidang yang berkaitan dengan pengolahan citra (gambar) seperti deteksi objek dan klasifikasi gambar, namun *DCNN* juga kadang digunakan untuk *natural language processing* seperti *speech processing*/*recognition* (konversi suara manusia ke dalam bentuk teks) dan sistem rekomendasi. (Choi et al, 2020)

Adapun beberapa kelebihan *DCNN* yaitu tidak memerlukan pengawasan manusia dalam proses mendeteksi fitur penting dari *input* secara otomatis, *DCNN* juga mengurangi jumlah parameter jaringan yang harus dilatih oleh karena sistem *weight sharing* (filter yang sama digunakan berulang kali untuk satu gambar) dan disaat bersamaan meningkatkan generalisasi sehingga mengurangi tingkat atau menghindari terjadinya *overfitting* pada jaringan. (Indolia et al, 2018)

Mengimplementasikan *DCNN* memiliki beberapa tantangan seperti untuk melatih *neural network* dibutuhkan dataset yang relatif besar, selanjutnya dari segi komputasional, modal yang dibutuhkan untuk melatih *DCNN* substansial, dan jika model yang digunakan terlalu kompleks dapat memicu terjadinya *overfitting*. (Indolia et al, 2018)

2.7 *Super-Resolution Convolutional Neural Network (SRCNN)* dan *Fast Super-Resolution Convolutional Neural Network (FSRCNN)*

SRCNN dan *FSRCNN* keduanya merupakan algoritma yang digunakan untuk meningkatkan kualitas gambar resolusi rendah/*Low Resolution* (*LR*) menjadi resolusi tinggi/*High Resolution (HR*). *FSRCNN* diusulkan oleh Chao Dong, Chen Change Loy, dan Xiaoou Tang untuk meningkatkan performa algoritma SRCNN agar dapat digunakan secara *real time*. (Dong et al, 2016)

Algoritma *SRCNN* dan *FSRCNN* memiliki perbedaan pada bagian arsitektur jaringan, algoritma *SRCNN* terdiri atas 3 bagian sedangkan *FSRCNN* terdiri atas 5 bagian. Struktur *SRCNN* terdiri atas bagian yang semuanya merupakan *layer* konvolusi, yang dapat dirincikan sebagai berikut :

1. *Patch extraction and representation* : *SRCNN* tidak langsung mengolah gambar secara keseluruhan melainkan *SRCNN* akan membagi gambar tersebut menjadi beberapa bagian kecil gambar yang saling tumpang tindih satu sama lain. *Layer* pertama ini dapat diekspresikan dengan operasi F_1 berikut ini :

$$F_1(Y) = max(0, W_1 * Y + B_1)$$

dimana W_1 merupakan filter dan B_1 merupakan bias, tanda '*' menggambarkan operasi konvolusi. Outputnya terdiri atas peta fitur n_1. Lalu untuk mendapatkan $F_1(Y)$ akan digunakan fungsi aktivasi *ReLu (Rectified Linear Unit),* dimana *ReLu* digunakan untuk mengatur nilai $F_1(Y)$ menjadi minimum 0 jika nilai bernilai negatif (kurang dari nol).

2. *Non-linear mapping* : Pada *layer* ini, *output* dari *layer* sebelumnya yaitu n_1 akan digunakan dan dipetakan ke dalam vektor dengan dimensi n_2 dengan menggunakan filter. Operasi pada *layer* ini dapat diekspresikan sebagai berikut :

$$F_2(Y) = max(0, W_2 * F_1(Y) + B_2)$$

sama seperti pada *layer* sebelumnya W_2 merupakan filter yang digunakan dalam operasi konvolusi dengan $F_1(Y)$ dan B_2 merupakan bias berdimensi n_2. Kemudian *ReLu* akan digunakan untuk mendapatkan *output* dari *layer* ini. Setiap *output* pada *layer* ini merupakan vektor dengan dimensi n_2 yang merupakan representasi bagian gambar beresolusi tinggi yang nanti

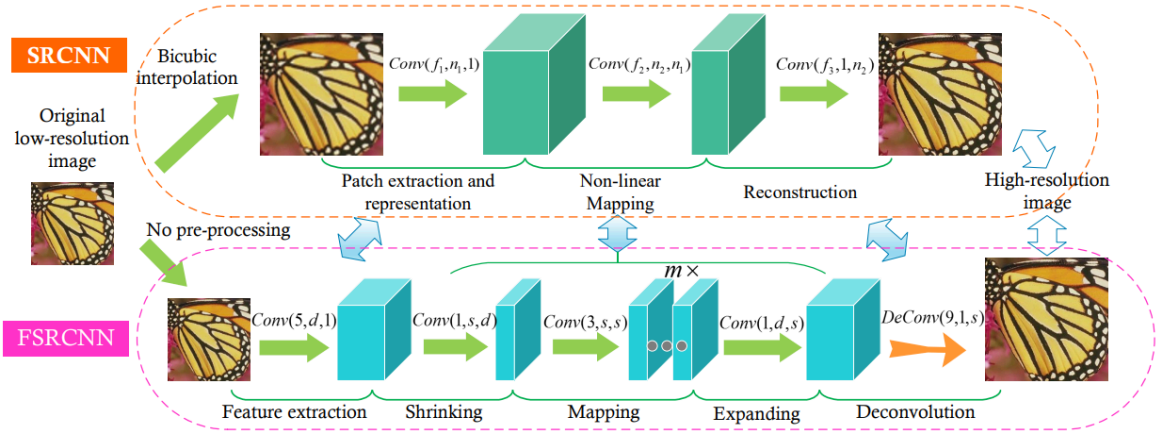
akan digunakan pada *layer* selanjutnya untuk merekonstruksi gambar dengan resolusi tinggi.

3. *Reconstruction* : Untuk mendapatkan gambar akhir (gambar lengkap/penuh) bagian-bagian gambar dengan resolusi tinggi yang saling tumpah tindih akan dirata-ratakan. Pemerataan tersebut dapat dilakukan dengan menggunakan filter W_3 pada operasi berikut ini :

$$F_3(Y) = W_3 * F_2(Y) + B_3$$

(Dong et al, 2015)

Seluruh bagian *SRCNN* merupakan operasi konvolusi/*layer* konvolusi, berbeda dengan *FSRCNN* yang terdiri atas 4 *layer* konvolusi dan 1 *layer deconvolution*. *SRCNN* memiliki tahapan awal yang mirip dengan *FSRCNN* namun selain perbedaan struktur *SRCNN* dan *FSRCNN* memiliki perbedaan yang terletak pada penggunaan gambar *input*, dimana *SRCNN* akan mengolah gambar original terlebih dahulu dengan menggunakan bicubic interpolation sedangkan *FSRCNN* akan menggunakan gambar original tanpa diolah terlebih dahulu dan langsung masuk pada tahapan *feature extraction* (mengekstraksi fitur-fitur penting) yang mirip dengan *bicubic interpolation* (Gambar 2.14). (Dong et al, 2015 dan 2016)



Gambar 2.14 Ilustrasi algoritma *SRCNN* dan *FSRCNN*

FSRCNN terdiri atas 5 bagian yaitu *feature extraction, shrinking, mapping, expanding,* dan *deconvolution*. Seluruh tahapan tersebut dapat dirincikan sebagai berikut :

1. *Feature extraction*

   Tahapan pertama pada *FSRCNN* mirip dengan tahapan pertama *SRCNN* namun *FSRCNN* tidak melakukan interpolasi pada gambar asli dan secara langsung melakukan ekstraksi fitur pada gambar *original*/asli. Pada *layer* konvolusional pertama dalam *SRCNN* kita dapat nyatakan sebagai berikut : $Conv(f_1, n_1, c_1)$ dimana f_1 merupakan ukuran filter, n_1 merupakan jumlah filter, dan c_1 merupakan jumlah channel. Dalam *SRCNN* ukuran filter yang digunakan pada tahapan pertama adalah 9, dalam *FSRCNN* ukuran filter yang digunakan adalah 5, sehingga nilai $f_1 = 5$, lalu dalam aspek jumlah *channel FSRCNN* dan *SRCNN* menggunakan jumlah yang sama yaitu 1, sehingga $c_1 = 1$, untuk nilai n_1 dapat dianggap sebagai dimensi fitur gambar resolusi rendah (*Low Resolution/LR*) dan dapat dilambangkan dengan d sehingga $n_1 = d$. Maka *layer* pertama *FSRCNN* dapat dilambangkan dengan notasi berikut : $Conv(5, d, 1)$

2. *Shrinking*

   Dalam *SRCNN*, setelah fitur dengan dimensi tinggi berhasil diekstraksi dari gambar resolusi rendah (*Low Resolution/LR*), fitur tersebut akan langsung dipetakan pada ruang fitur gambar resolusi tinggi (*High Resolution/HR*). Namun, dalam memetakan fitur *LR* yang dilambangkan dengan d memiliki kompleksitas komputasi yang tinggi dikarenakan dimensi fitur d biasanya memiliki ukuran yang sangat besar. Untuk mengurangi biaya komputasi tersebut, *FSRCNN* akan menggunakan *layer* 1 x 1 untuk mengurangi ukuran fitur *LR* dengan dimensi d, sehingga ukuran filter pada *layer* ini dapat memiliki nilai tetap sebagai berikut $f_2 = 1$. Filter akan bekerja seperti kombinasi linear dalam fitur *LR* sehingga dimensi fitur berkurang dari d menjadi dimensi fitur yang lebih kecil yang dapat dilambangkan dengan s ($n_2 = s << d$). *Layer* ini secara keseluruhan dapat dinotasikan sebagai berikut : $Conv(1, s, d)$

3. *Mapping*

   Ada beberapa faktor dalam tahapan *non-linear mapping* yang mempengaruhi performa *SR (Super Resolution)*, dimana ada dua faktor yang paling berpengaruh yaitu *width* (jumlah filter dalam sebuah *layer*)

dan *depth* (jumlah *layer*). Untuk menyeimbangkan ukuran jaringan dan performa, dalam *FSRCNN* digunakan filter dengan ukuran 3 (f_3 = 3), dan digunakan beberapa filter 3 x 3 dan tidak menggunakan satu filter yang berukuran besar. Selanjutnya seluruh *layer* mapping memiliki jumlah filter yang sama yaitu $n_3 = s$, dengan tujuan menjaga konsistensi. Jumlah mapping *layer* yang menentukan akurasi dan kompleksitas dari tahapan mapping ini dapat dilambangkan sebagai *m*, sehingga tahapan ini dapat dinotasikan sebagai berikut : *m * Conv(3, s, s)*

4. *Expanding*

   Expanding merupakan kebalikan dari proses *shrinking*. Pada proses *shrinking*, untuk meningkatkan efisiensi komputasi, jumlah dimensi fitur *LR* dikurangi. Namun, apabila gambar *HR* dibentuk berdasarkan fitur berdimensi rendah, gambar yang akan dihasilkan akan memiliki kualitas yang rendah. Maka sebab itu pada tahapan *expanding* ini dimensi fitur *HR* akan dikembangkan dengan cara melakukan operasi konvolusi menggunakan beberapa filter 1 x 1, dimana jumlah filter ini sesuai dengan jumlah filter pada *layer feature extraction* yaitu *d*. Tahapan *expanding* ini dapat dinotasikan sebagai berikut : *Conv(1, d, s)*

5. *Deconvolution*

   Tahapan terakhir yaitu *deconvolution*, dimana dalam *layer* ini fitur-fitur yang didapatkan pada *layer* sebelumnya digabungkan kemudian resolusi ditingkatkan dengan menggunakan filter-filter *deconvolution*. Operasi deconvolution merupakan kebalikan dari konvolusi. Pada operasi konvolusi, dengan menggunakan parameter "*stride (k*)" (parameter dalam *FSRCNN* yang menentukan seberapa besar penggesaran filter terhadap gambar, misalkan *k* = 1 maka *filter* akan bergeser sebesar 1 *pixel* ke bagian gambar selanjutnya), filter dikonvolusi dengan gambar sesuai nilai *k*, maka akan menghasilkan *output* 1/*k* dari *input*. Oleh karena operasi dekonvolusi merupakan kebalikan dari konvolusi, posisi *input* dan *output* dari operasi konvolusi dibalik sehingga *output*nya menjadi *k* kali *input*. Berdasarkan *output* tersebut, jika nilai *k* ditentukan nilainya yaitu *k* = *n*, dimana *n* merupakan faktor *upscaling* (seberapa besar peningkatan kualitas gambar),

dengan demikian *output* yang langsung didapatkan adalah gambar resolusi tinggi (*HR*) yang direkonstruksi sesuai dengan faktor *upscaling* n (misalkan *n* bernilai dua maka gambar yang direkonstruksi akan menjadi dua kali lebih besar atau resolusinya akan meningkat 2 kali lipat). *Layer deconvolution* ini dapat dinotasikan sebagai berikut : *DeConv(9, 1, d)*

(Dong et al, 2016)

2.8 Peak Signal-to-Noise Ratio (*PSNR*)

PSNR merupakan metrik yang digunakan untuk mengukur kualitas gambar yang diolah dibandingkan dengan gambar asli (*original*) melalui perhitungan rasio dalam desibel. Nilai *PSNR* ini dapat diukur dengan menggunakan berbagai aplikasi, salah satu aplikasi yang dapat digunakan yaitu *ffmpeg*. Nilai *PSNR* berbanding lurus dengan kualitas gambar hasil olahan, semakin tinggi nilai *PSNR* maka semakin baik kualitas gambar tersebut. (Salomon, 2007). Nilai *PSNR* dalam decibel (dB) didapatkan dengan menggunakan formula berikut :

$$PSNR = 10\, log_{10}(\frac{MAX^2}{MSE}) \qquad (2.1)$$

(*ffmpeg*, n,d,)

Dimana *MAX* merupakan nilai maksimum dari sebuah *pixel* untuk komponen tertentu, dan *MSE (Mean Square Error)* merupakan perbedaan rata-rata antar *pixel* seluruh gambar, sehingga semakin tinggi nilai *MSE* maka gambar asli dan gambar olahan semakin berbeda. Untuk perhitungan *MSE* dapat menggunakan persamaan berikut :

$$MSE = \frac{1}{N}\Sigma\Sigma(E_{ij} - o_{ij})^2 \qquad (2.2)$$

(Salomon, 2007)

N merupakan total jumlah *pixel* (ukuran gambar), E merupakan gambar tepi, dan o adalah gambar asli. (Salomon, 2007)

Range nilai *PSNR* pada umumnya berkisaran antara 30 dB sampai dengan 40 dB untuk video 8 bit dengan kualitas bagus, dan lebih dari 40 dB untuk kualitas gambar sangat bagus. (Chervyakov et al, 2020)

2.9 Hukum Amdahl

Pada tahun 1967, seorang bernama Gene Amdahl menemukan ada sebuah tantangan dalam paralelisasi untuk meningkatkan performa komputasi, hal ini terjadi karena dalam mengeksekusi sebuah program ada bagian program yang tidak dapat dieksekusi secara paralel dan cenderung harus dieksekusi secara serial, sehingga bagian kode tersebut tidak mendapatkan manfaat dari pemrosesan paralel. Amdahl menemukan bahwa meningkatkan paralelisasi dari sebuah sistem sebesar *N* (dimana N merupakan jumlah prosesor atau *core* yang tersedia dalam sebuah sistem) tidak akan pernah mencapai performa sebanyak faktor *N*, hal ini disebabkan oleh bagian kode yang tidak dapat dieksekusi secara paralel (harus dieksekusi secara serial) dan juga faktor-faktor *parallelization overheads* (faktor-faktor seperti membuat dan menghilangkan *thread*, mengunci data agar nilainya tidak diubah secara bersamaan oleh beberapa *thread* sekaligus, dan sinkronisasi perhitungan yang dilakukan oleh masing-masing *thread* yang tersedia). Selanjutnya untuk menjelaskan hubungan tersebut secara matematis, Amdahl menciptakan hukum yang disebut dengan hukum Amdahl. Hukum Amdahl adalah hukum yang digunakan untuk mengukur secara teoritis seberapa cepat peningkatakan waktu eksekusi sebuah aplikasi yang diimplementasikan komputasi paralel dengan jumlah unit komputasi tertentu (processor) atau dalam konteks *multithreading* unit komputasi yang digunakan adalah *thread*.

Peningkatan waktu eksekusi (*speedup*) teoritis sebuah aplikasi berdasarkan hukum Amdahl dapat dihitung dengan menggunakan persamaan 2.3 berikut (Gillespie dan Intel, n.d.) :

$$speedup = \frac{1}{S+\frac{1-S}{N}} - O_n \qquad (2.3)$$

Dimana :

- S = (waktu total eksekusi program - p) / waktu total eksekusi program , dimana p = Waktu yang dibutuhkan untuk menjalankan bagian kode yang dapat diparalelkan dan S merupakan rasio beban kerja bagian kode yang dijalankan secara serial (nilainya terletak antara 0 sampai dengan 1),
- N = jumlah prosesor/*thread*

- O_n = *parallelization overhead* (faktor-faktor lain yang dapat memperlambat eksekusi program paralel seperti sinkronisasi data)

(Gillespie dan Intel, n.d.)

2.10 SSIM (*Structural Similarity Index Measure*)

SSIM (*Structural Similarity Index Measure*) merupakan metrik yang digunakan untuk membandingkan dua gambar dengan membandingkan tiga faktor yaitu iluminasi (*luminance*), kontras, dan struktur gambar. SSIM dapat didefinisikan berdasarkan rumus berikut :

$$SSIM(f,g) = l(f,g)\, c(f,g)\, s(f,g) \quad (2.4)$$

Dimana :

- Rentang nilai $SSIM(f,g)$ berkisar dari 0 sampai 1, dimana jika nilai SSIM mendekati 0 artinya tidak ada kemiripan antara dua gambar dan semakin mendekati 1 kedua gambar tersebut semakin mirip dan jika SSIM bernilai 1 artinya kedua gambar mirip sempurna atau identikal
- $l(f,g)$ merupakan fungsi perbandingan iluminasi (*luminance*) antara dua gambar
- $c(f,g)$ merupakan fungsi perbandingan kontras antara dua gambar
- $s(f,g)$ merupakan fungsi perbandingan struktur dua gambar

Selanjutnya fungsi perbandingan iluminasi antara dua gambar dapat didefinisikan dengan persamaan berikut :

$$l(f,g) = \frac{2\mu_f\mu_g + C_1}{\mu_f^{\,2} + \mu_g^{\,2} + C_1} \quad (2.5)$$

Persamaan (2.5) merupakan fungsi perbandingan iluminasi dimana persamaan ini mengukur kedekatan iluminasi antara dua gambar, sehingga untuk mencapai faktor iluminasi maksimum nilai μ_f harus sama dengan μ_g.

Lalu, fungsi perbandingan kontras antara dua gambar didefinisikan dengan persamaan berikut :

$$c(f,g) = \frac{2\sigma_f\sigma_g + C_2}{\sigma_f^{\,2} + \sigma_g^{\,2} + C_2} \quad (2.6)$$

Persamaan (2.6) merupakan fungsi perbandingan kontras dimana persamaan ini mengukur kedekatan kontras antara dua gambar, pada fungsi ini, kontras diukur berdasarkan deviasi σ_f dan σ_g. Untuk mencapai faktor kontras maksimum nilai σ_f harus sama dengan σ_g.

Dan fungsi perbandingan struktur dua gambar dapat didefinisikan dengan persamaan berikut :

$$s(f,g) = \frac{\sigma_{fg}+C_3}{\sigma_f\sigma_g+C_3} \qquad (2.7)$$

Persamaan (2.7) merupakan fungsi perbandingan struktur dimana persamaan ini mengukur koefisien korelasi antara gambar f dan gambar g, dimana σ_{fg} merupakan kovarian antara f dan g. (Hore et al., 2010)