3. ANALISIS DAN DESAIN SISTEM

3.1 Analisis Masalah

Untuk menguji kemampuan dari *motion vector* atau *optical flow* dengan model CNN-LSTM dalam melakukan rekognisi aktivitas manusia, diperlukan suatu sistem yang dapat melakukan rekognisi aktivitas manusia dari *input* video dengan *encoding* H.264.

Data *motion vector* yang digunakan untuk melakukan rekognisi ter-*encode* di-dalam video, sehingga diperlukan *decoder* untuk men-*decode* data *frame* RGB dan *list motion vector* mentahan. Proses *decode* pada video dilakukan pada setiap *frame*, jika *input* dari *decoder* merupakan video *stream* (seperti *stream* CCTV), proses *decoder* yang terhambat oleh proses yang lebih lambat dari pada *frame rate* dari *input* video (seperti proses *preprocessing* yang berat) akan menimbulkan *delay*. Untuk mengatasi *delay* ini, proses *decoder* akan dibuat sebagai proses yang independen dari proses utama.

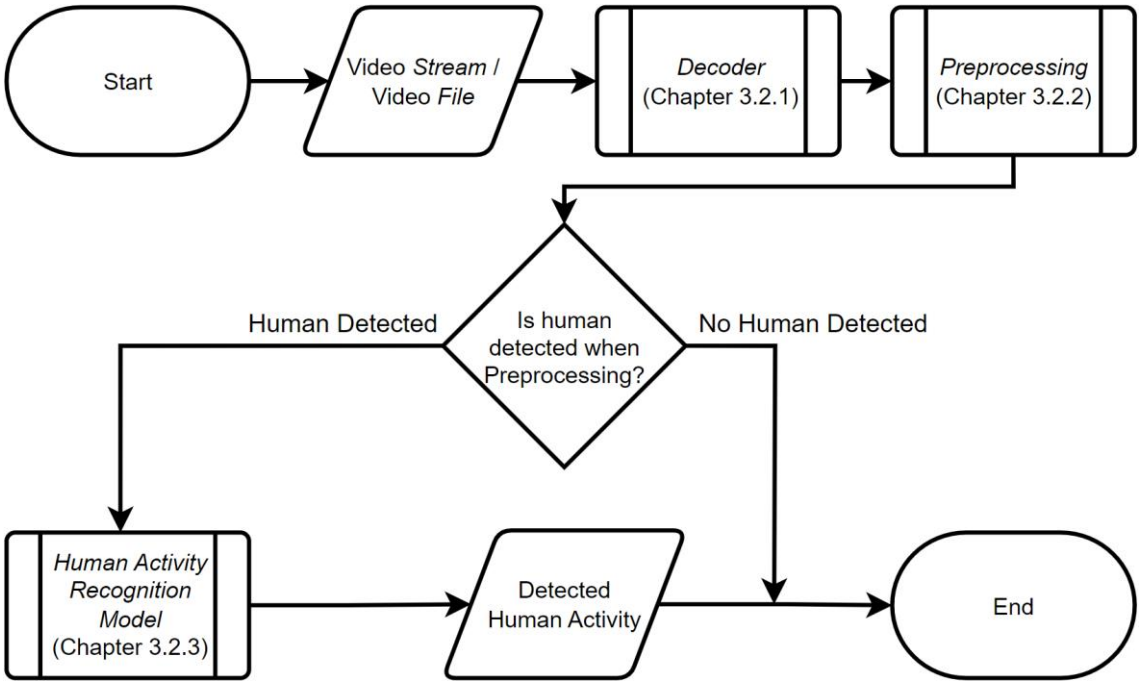
Output dari *decoder* tidak dapat langsung dipakai karena model rekognisi aktivitas manusia yang akan digunakan menerima *input* berupa *frame* data pergerakan dengan resolusi 224x224, sehingga diperlukan suatu sistem *preprocessing* yang akan mempersiapkan data sebelum di-*input*-kan ke model rekognisi aktivitas manusia. Pada metode yang menggunakan *motion vector* diperlukan proses konversi data yang bentuk awalnya berupa *list* dari *motion vector* menjadi bentuk *frame* dari *motion vector*. Pada metode yang menggunakan *optical flow*, diperlukan proses untuk melakukan generasi *frame optical flow* menggunakan model *optical flow FlowFormer*. *Frame* data pergerakan yang dihasilkan pada metode *motion vector* maupun *optical flow* memiliki *noise* dan *empty space* atau data pergerakan yang tidak relevan (bukan manusia), oleh karena itu, *frame-frame* data pergerakan ini akan melalui proses *masking* dan *cropping* menggunakan *segmentation mask* dan *bounding box* yang dihasilkan oleh model YOLOv7 *Mask* untuk mengeliminasi *noise* dan data-data tidak relevan lainnya. Pada proses training, proses *preprocessing* akan dijalankan pada video yang sebelumnya telah dijalankan *preprocessing*. Meskipun *preprocessing* didesain untuk berjalan dengan cepat, proses ini tetap saja memakan waktu dan tenaga komputasi. Untuk menghindari *preprocessing* yang berulang ini, maka diperlukan sistem penyimpanan dan sistem *load* hasil *preprocessing* yang dapat membantu mengurangi durasi *training* dan *tuning* model.

Sejumlah *frame-frame* data pergerakan hasil *preprocessing* akan menjadi *input* dari model rekognisi aktivitas manusia yang terdiri dari bagian CNN dan LSTM. Model ini akan

menghasilkan *output* berupa aktivitas manusia yang telah di-rekognisi. Model ini akan di-*train* mengunakan dataset yang dijelaskan pada bab 2.1.8, tetapi video pada dataset ini memiliki tipe *encoding* MPEG-4 yang merupakan *encoding* kuno dan tidak populer digunakan dalam media video dan *streaming* (seperti CCTV) yang menggunakan *encoding* H.264, maka video-video pada dataset perlu dikonversikan terlebih dahulu ke *encoding* H.264.

3.2 Desain Sistem

Program rekognisi aktivitas manusia pada skripsi ini akan terdiri dari tiga bagian utama, bagian pertama adalah bagian *capturing* video dari kamera atau *reading* file video, kedua adalah melakukan *preprocessing* untuk mendapatkan data *masking* untuk mengeliminasi *noise* dan melakukan *cropping* pada data *motion* vector atau *optical flow* dan yang terakhir adalah bagian rekognisi yang dilakukan oleh model rekognisi. *Output* dari proses *masking* akan menentukan jika model rekognisi perlu dijalankan. Alur besar dari program rekognisi dapat dilihat pada Gambar 3.1.



Gambar 3.1 *Flowchart* Proses Rekognisi Aktivitas Manusia

3.2.1 *Decoder*

Decoder merupakan suatu proses untuk melakukan *decoding stream* atau video H.264 untuk mendapatkan data *frame* dan *motion vector*. Pada proses ini, program *decoder* akan menghasilkan data *frame* RGB dan *list* dari *motion vector* mentahan yang akan diubah menjadi satu *frame* dari *motion vector* pada proses *preprocessing*. Kecepatan dari proses *decoding* ini akan ter-*limit* oleh faktor kecepatan media penyimpanan (SSD, HDD, dsb) atau kecepatan internet untuk *stream video* dan kecepatan dari perangkat *decoder*. Untuk *stream* video akan digunakan protokol RTSP (*Real Time Streaming Protocol*). Detail *output* dari *decoder* dijelaskan pada Tabel 3.1.

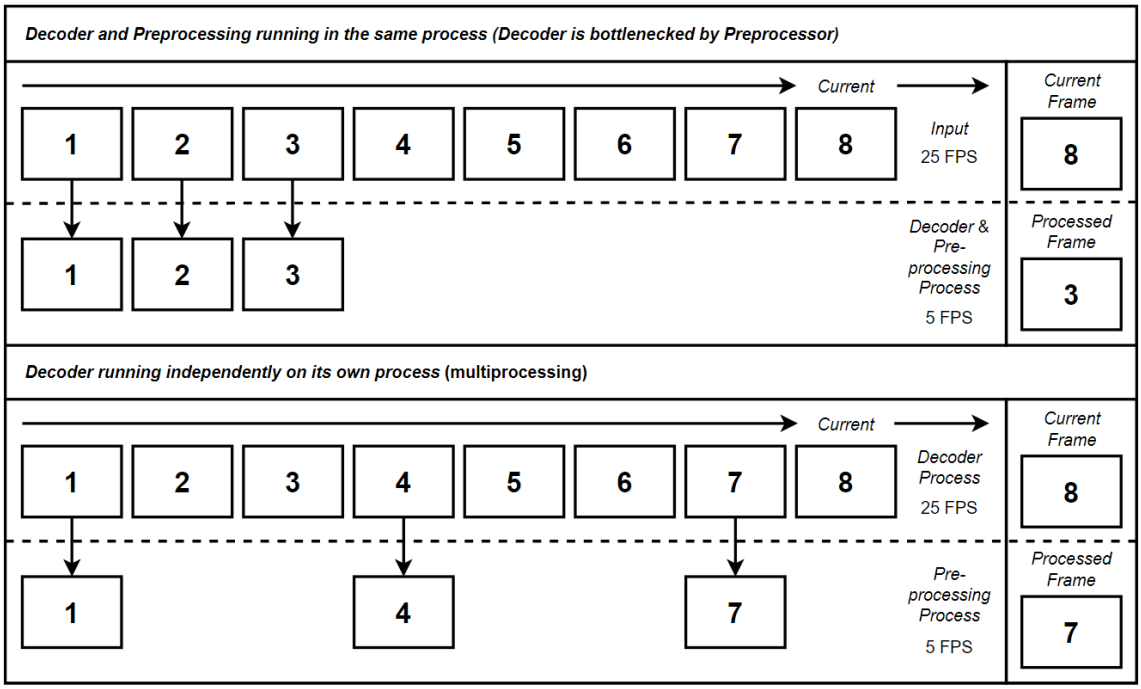
Tabel 3.1

Detail Struktur Data *Output* dari *Decoder*

INDEX	***NAME***	***TYPE***	***DETAILS***
Decoder Output			
0	*Success*	bool	*True* apabila *motion vector* ditemukan dan berhasil di-*decode*, jika *False* maka *decode* untuk *frame* gagal dan nilai *Frame* RGB dan *Raw Motion Vector List* akan menjadi kosong.
1	*Frame RGB*	numpy.ndarray[uint8]	*Array* dengan ukuran HxWxC yang merupakan *frame* BGR (3 *channel*) dengan resolusi WxH. Jika *decoding* gagal, maka *shape* dari *array* ini (0, 0, 3).
2	*Raw Motion Vector List*	List[numpy.ndarray[Any]]	*List* dari *Raw Motion Vector*. Jika *decoding* gagal, maka akan menjadi *list* kosong. Detail dari *Raw Motion Vector* dibahas pada bagian selanjutnya.
Raw Motion Vector Detail			
0	*source*	int32	*Offset* dari *reference frame* dari *motion vector*. Jika *source* < 0, *reference frame* berasal dari *frame* sebelum, dan sebaliknya jika *source* > 0 maka *reference frame* berasal dari *frame* sesudah.

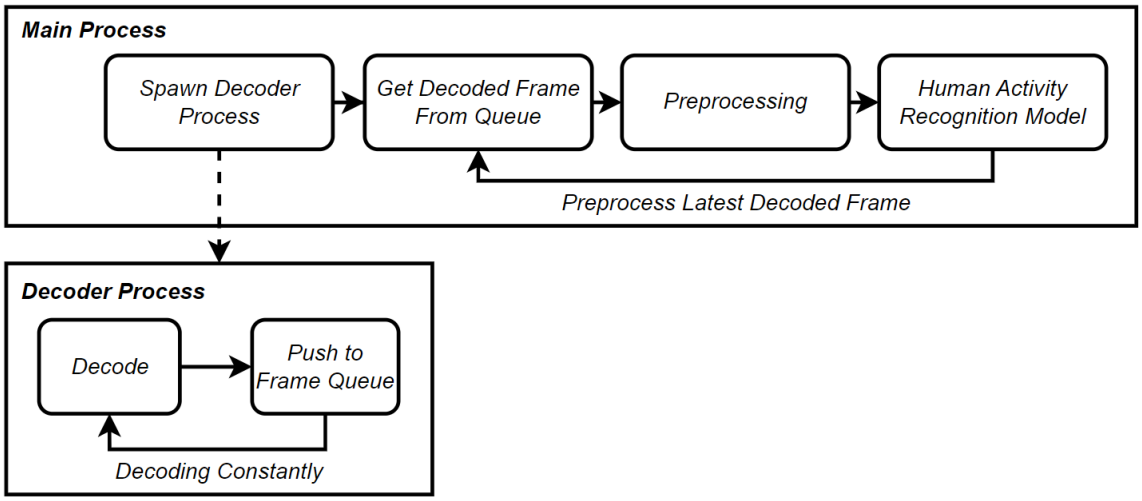
INDEX	***NAME***	***TYPE***	***DETAILS***
1	w	uint8	Panjang *macroblock* dari *motion vector*.
2	h	uint8	Lebar *macroblock* dari *motion vector*.
3	src_x	int16	Posisi tengah X-*Axis* absolut yang merupakan destinasi *motion vector* pada *reference frame*.
4	src_y	int16	Posisi tengah Y-*Axis* absolut yang merupakan destinasi *motion vector* pada *reference frame*.
5	dst_x	int16	Posisi tengah X-*Axis* absolut yang merupakan sumber *motion vector* pada *current frame*.
6	dst_y	int16	Posisi tengah Y-*Axis* absolut yang merupakan sumber *motion vector* pada *current frame*.
7	*motion_x*	int32	Bagian X-Axis dari *motion vector* (*motion_scale* * (src_x – dst_x)).
8	*motion_y*	int32	Bagian Y-Axis dari *motion vector* (*motion_scale* * (src_y – dst_y)).
9	*motion_scale*	uint16	Skala *motion vector* yang digunakan pada *motion_x* dan *motion_y*. Skala digunakan untuk menaikan skala *motion vector* dari bentuk *float* ke bentuk *integer*.

Pada proses *reading* video dataset, setiap *frame* akan di-*decode* satu-persatu dan diproses per-*frame*-nya, tetapi hal ini tidak baik untuk *stream* CCTV *realtime* karena ketika proses lebih lama dari *frame rate stream* maka akan terjadi *delay* antara *frame* yang diproses dan dengan *frame* yang terbaru dan *delay* tersebut akan terus bertambah.



Gambar 3.2 Ilustrasi *Delay* pada *Decoder* dan *Preprocessing*

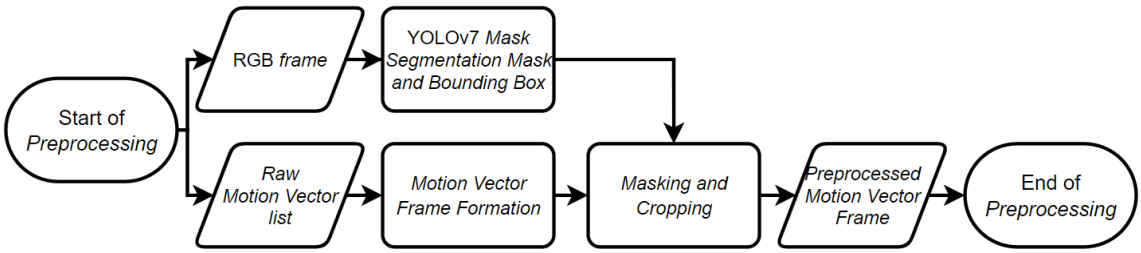
Untuk mengatasi *delay* pada *stream realtime* seperti pada Gambar 3.2, *decoder* perlu berjalan secara independen dari proses utama agar *decoder* selalu berjalan berusaha men-*decode frame* terbaru dan proses selanjutnya selalu mendapatkan *frame* terbaru. Proses yang diilustrasikan pada Gambar 3.3 akan diimplementasikan dengan menggunakan sistem *multiprocessing* di mana proses utama akan meng-*spawn decoder* yang akan berjalan dalam loop *decoding* secepat mungkin dan *frame* terbaru akan di-*push* ke *queue* yang menghubungkan proses utama dan proses anakan *decoder*. Karena proses *decoding* ini ringan dan berjalan di-*hardware*, maka proses independen ini tidak mempengaruhi performa proses yang lain.



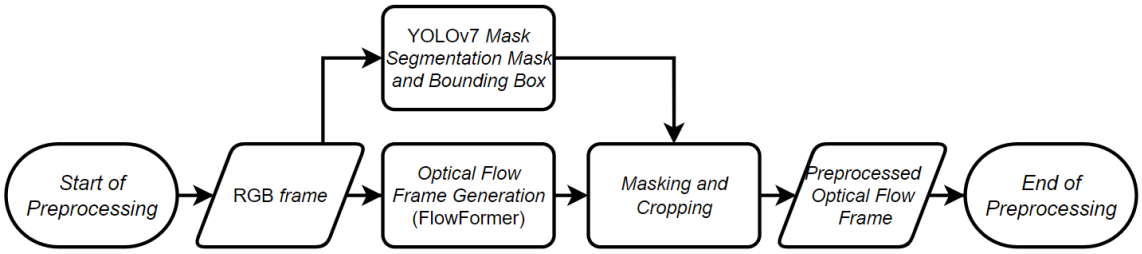
Gambar 3.3 *Block Diagram* Detail *Multiprocessing* pada *Decoder*

3.2.2 *Preprocessing*

Proses *preprocesing* merupakan bagian dari proses utama yang dijalankan sebelum melakukan proses rekognisi. Proses ini terdiri dari dua bagian, pertama adalah pembentukan *frame* data dari suatu pergerakan, pembentukan ini dapat menggunakan proses *motion vector* atau menggunakan proses *optical flow*. Setelah itu *frame* data RGB akan digunakan untuk mendapatkan data *masking* dan *bounding box* yang digunakan untuk melakukan *masking* dan *cropping* pada *frame* data dari suatu pergerakan (*motion vector* atau *optical flow*) untuk mengeliminasi *noise* dan data tidak relevan (pergerakan bukan manusia). *Frame-frame* data pergerakan yang telah diproses ini akan masuk kedalam suatu *queue* yang akan menyimpan n-*frame* sebelumnya untuk diinputkan ke model rekognisi. Proses *preprocessing* untuk metode *motion vector* dan untuk metode *optical flow* memiliki perbedaan sedikit yang ditunjukkan pada Gambar 3.4 dan Gambar 3.5.



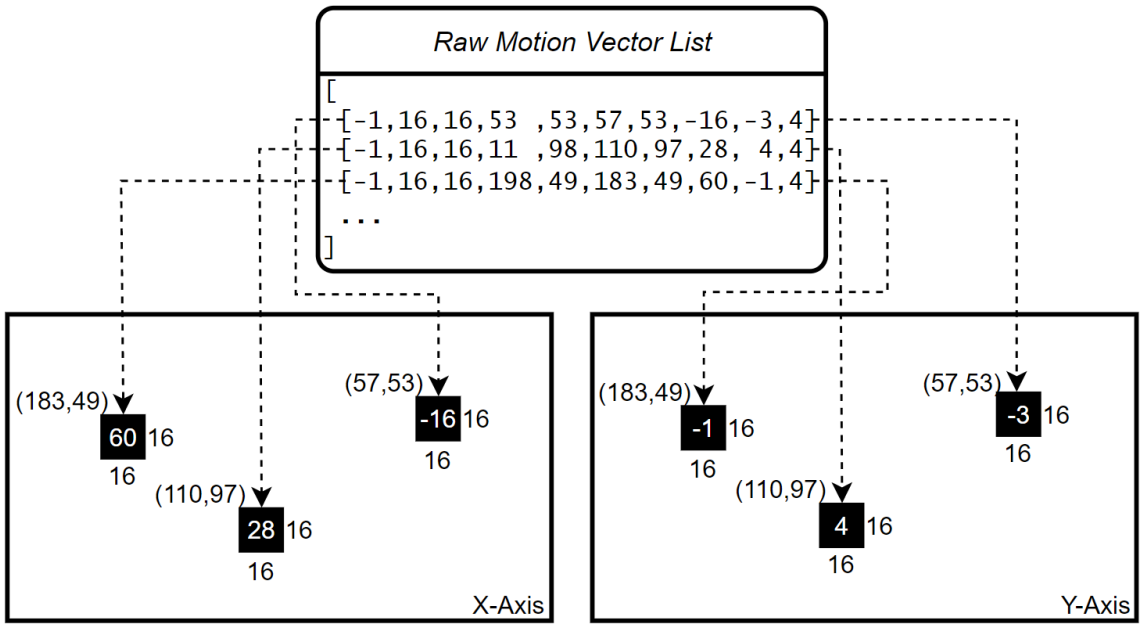
Gambar 3.4 *Flowchart Preprocessing* Metode *Motion Vector*



Gambar 3.5 *Flowchart Preprocessing* Metode *Optical Flow*

3.2.2.1 Proses Pembentukan *Frame Motion Vector*

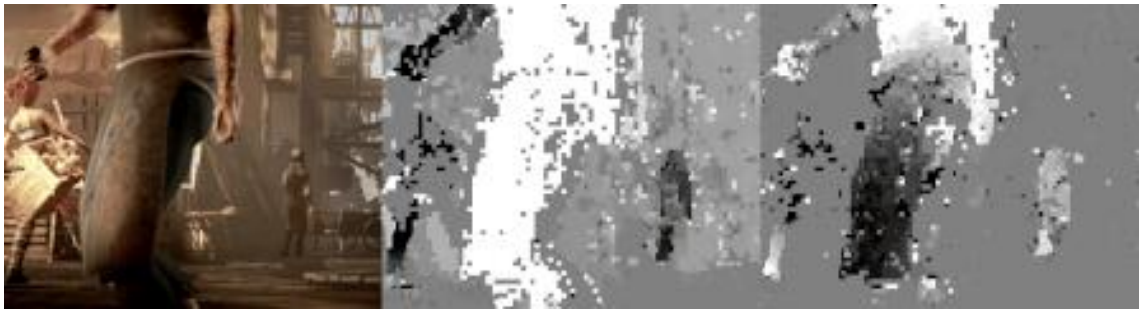
Pada proses *motion vector* akan dilakukan pembentukan *frame* dari *motion vector* dari yang awalnya merupakan *list motion vector* mentahan (*raw motion vector list*) seperti pada Gambar 3.6. Resolusi dari *frame* dari *motion vector* ini akan mengikuti resolusi *frame* RGB aslinya. *Motion vector* yang digunakan hanya *forward motion vector* yang menggambarkan pergerakan *pixel* dari *frame* sebelum ke *frame* sesudah. Jika suatu *frame* video tidak memiliki *motion vector*, maka *frame* dari *motion vector* dari *frame* terakhir akan digunakan.



Gambar 3.6 Pembentukan *Frame Motion Vector* dari *Raw Motion Vector List*

Akan ada dua *frame* dari *motion vector* yang dihasilkan, *frame* yang menggambarkan pergerakan horizontal (x-*axis*) dan *frame* yang menggambarkan pergerakan vertikal (y-*axis*) seperti pada Gambar 3.7. Setelah itu *frame-frame* ini akan dilakukan *clipping* dengan *limit bounding* dan *rescale* agar *frame motion vector* memiliki nilai antara 0 sampai 255. *Limit*

bounding kecil akan memperkuat sinyal gerakan dengan membuang gerakan yang terlalu besar. Perbandingan nilai *limit bounding* pada frame *motion vector* terlihat pada Gambar 3.8.



Gambar 3.7 X-*Axis* dan Y-*Axis* pada *Frame Motion Vector*

Gambar 3.8 *Bounding* pada *Frame Motion Vector*, *Limit Bounding* Kiri ke Kanan (32, 64, 128, 256, 512)

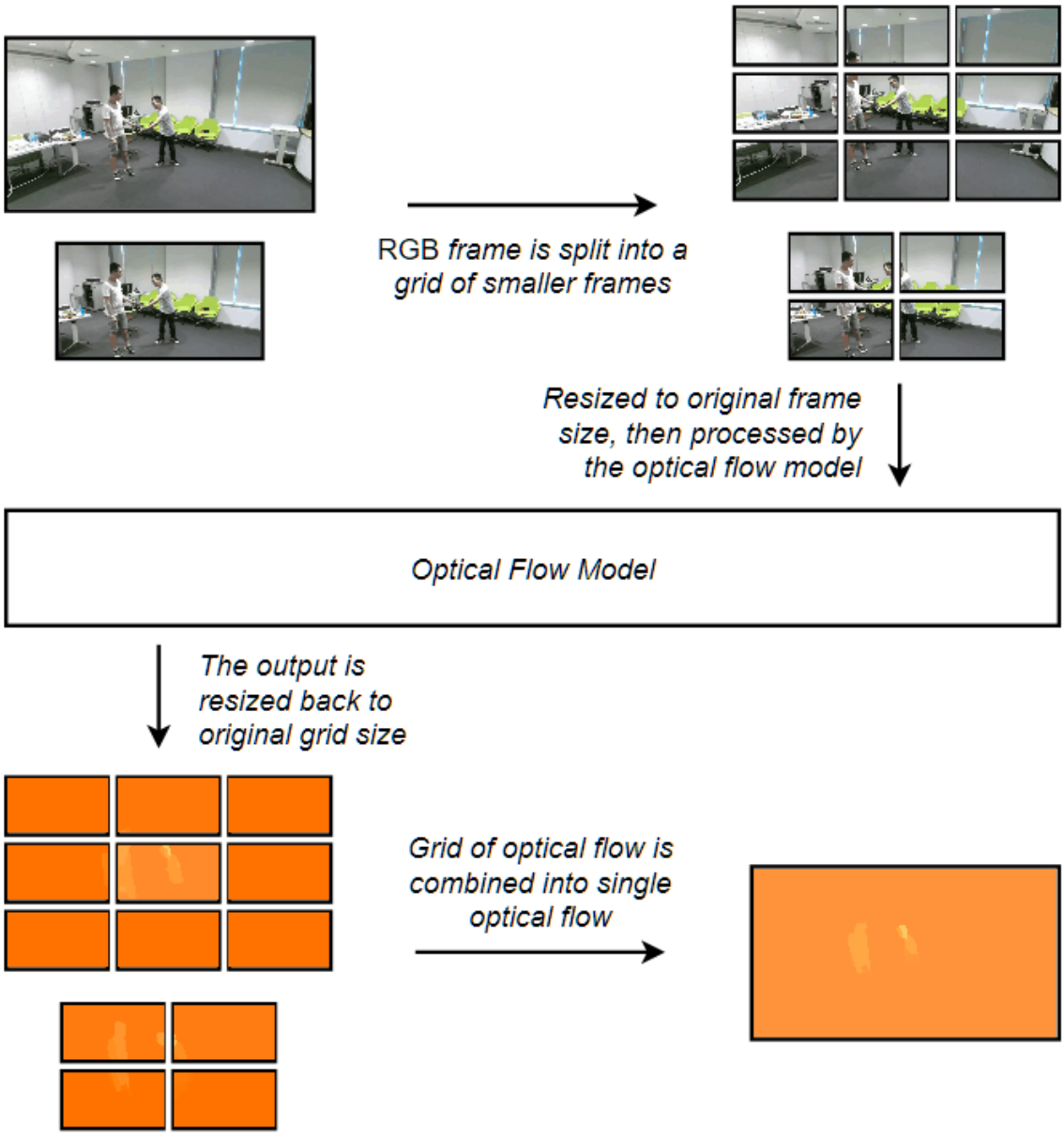
3.2.2.2 Proses Generasi *Optical Flow* (Menggunakan Model *FlowFormer*)

Pada proses *optical flow*, dua *frame* RGB akan digunakan untuk menghasilkan *frame optical flow* menggunakan model *FlowFormer pretrained*. Output *frame optical flow* adalah dua *frame* data pergerakan yang salah satunya menggambarkan pergerakan horizontal (x-*axis*) dan *frame* yang lain menggambarkan pergerakan vertikal (y-*axis*) seperti pada Gambar 3.9. *Frame-frame optical flow* yang sudah dihasilkan akan dilakukan *clipping* dengan *limit bounding* dan *rescale* agar *frame optical flow* memiliki nilai antara 0 sampai 255. *Limit bounding* kecil akan memperkuat sinyal gerakan dengan membuang gerakan yang terlalu besar. Perbandingan nilai *limit bounding* pada *frame optical flow* terlihat pada Gambar 3.10.

Gambar 3.9 X-*Axis* dan Y-*Axis* pada *Frame Optical Flow*

Gambar 3.10 *Bounding* pada *Frame Optical Flow*, *Limit Bounding* Kiri ke Kanan (32, 64, 128, 256, 512)

Optical flow menghasilkan *frame* data pergerakan yang sangat jernih dan tajam, tetapi *optical flow* memiliki kelemahan. *Optical flow* tidak dapat dengan baik mendeteksi gerakan kecil atau pergerakan pada objek kecil. Untuk mengatasi hal ini, diimplementasikan fungsi *grid sampling*. *Grid sampling* seperti yang diilustrasikan pada Gambar 3.11, memecah *frame* RGB menjadi beberapa bagian, lalu pecahan *frame* RGB ini di-*upscale* ke resolusi aslinya yang selanjutnya dimasukkan ke model *optical flow*. Lalu output dari model *optical flow* digabung kembali dengan membandingkan gerakan absolut dan memilih nilai yang terbesar. Tujuan *grid sampling* adalah untuk memperkuat efek gerakan kecil sehingga gerakan kecil terlihat seperti gerakan besar pada model *optical flow*. Dengan ini gerakan kecil dapat lebih terlihat, tetapi hasil *optical flow* menjadi lebih *noisy* dan menjadi lebih lambat.



Gambar 3.11 Ilustrasi Proses *Grid Sampling* (3x3)

Gambar 3.12 Perbandingan Sebelum dan Sesudah memakai *Grid Sampling* (3x3)

Tabel 3.2

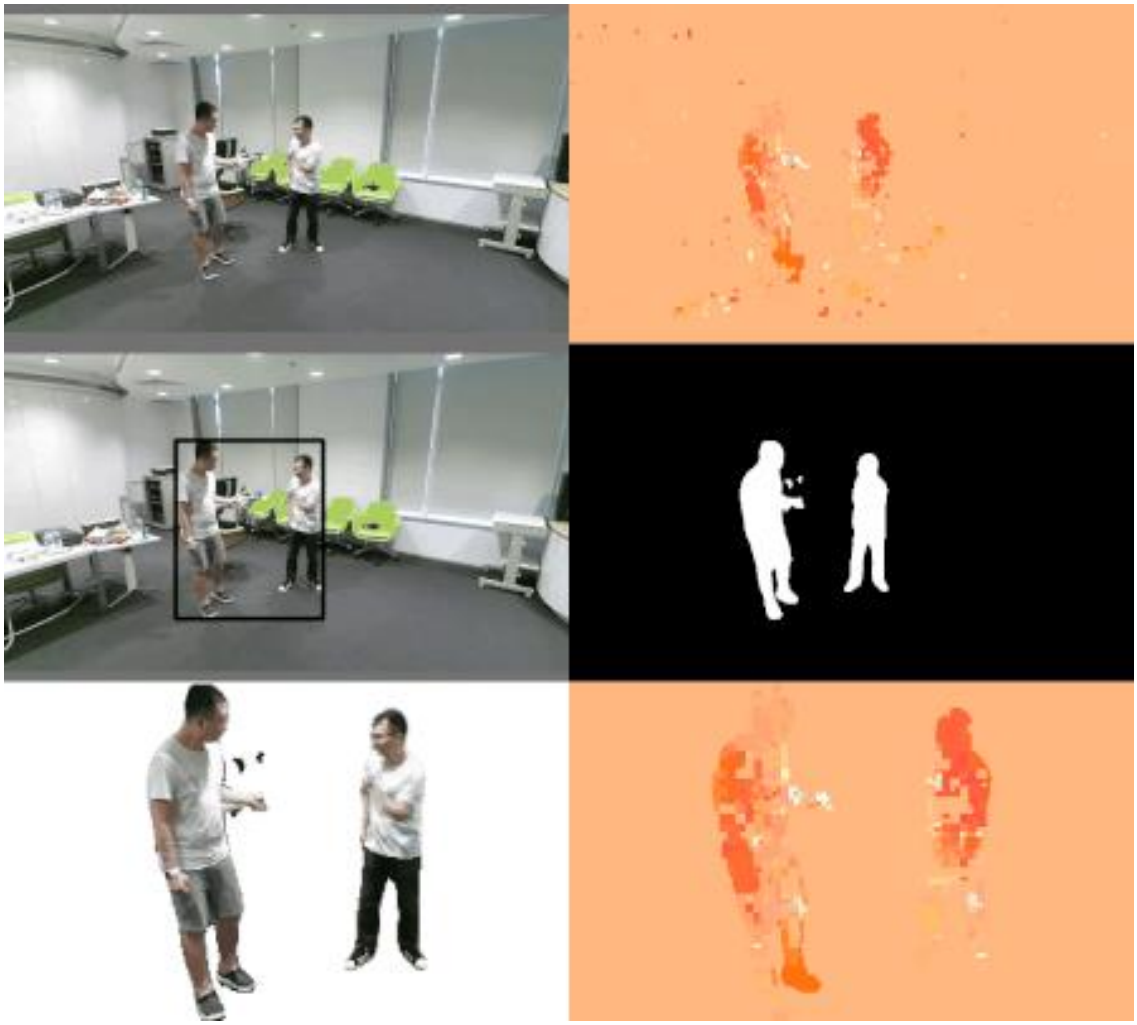
FPS dari Tidak Menggunakan dan dengan Menggunakan *Grid Sampling* (3x3)

RESOLUTION	***NO GRID SAMPLING* (FPS)**	***GRID SAMPLING* (FPS)**
960p	8.87 FPS	2.58 FPS
640p	12.41 FPS	4.94 FPS
320p	20.65 FPS	7.04 FPS

Seperti yang dapat dilihat pada Tabel 3.2, penggunakan *grid sampling* akan memperlambat output dari *optical flow*, tetapi seperti yang dapat dilihat pada Gambar 3.12 *grid sampling* membantu untuk meningkatkan sensitivitas agar sebanding dengan *motion vector*, maka pada skripsi ini *grid sampling* akan digunakan.

3.2.2.3 Proses *Masking* dan *Cropping* (Menggunakan Model YOLOv7 *Mask*)

Pada proses *masking dan cropping*, model YOLOv7 *Mask* yang sudah di-*pretrain* digunakan untuk mendapatkan *mask* objek manusia. *Mask* ini akan digunakan untuk melakukan eliminasi *noise* dan eliminasi data yang tidak relevan pada *frame motion vector* atau *optical flow*. Selain data *mask*, YOLOv7 *Mask* juga menghasilkan *bounding box* untuk setiap manusia yang terdeteksi. *Bounding box* yang paling mendekati *center* dari *frame* dipilih menjadi *bounding box* utama dan *bounding box* yang jaraknya *center*-nya dibawah atau sama dengan *threshold* yang ditentukan akan digabungkan menjadi satu untuk mendapatkan *bounding box* dari ROI (*Region of Interest*) yang digunakan untuk melakukan *cropping*. Contoh dari *masking* dan *cropping* dapat dilihat pada Gambar 3.13.



Gambar 3.13 Contoh Proses *Masking* dan *Cropping* menggunakan *Bounding Box* dengan YOLOv7 *Mask*

YOLOv7 *Mask* sudah cukup cepat untuk model *instance* segmentation, tetapi masih kurang cepat untuk kebutuhan *realtime*. Untuk mengatasi hal ini, dilakukan modifikasi kecil ke fungsi *inference* pada model *pretrained* untuk menghilangkan *dynamic branches* sehingga dapat di-*compile* sesuai dengan resolusi *input*. *Compile* model dilakukan agar model lebih teroptimisasi terhadap resolusi *input* dan *hardware* di mana model tersebut dijalankan. Proses *compile* ini memakan waktu yang sangat lama, karena itu hasil *compile* akan disimpan sehingga jika model dijalankan dengan resolusi *input* dan *hardware* yang sudah pernah digunakan, model akan secara otomatis *load* model yang sudah ter-*compile*. Hasil *compile* merupakan model statis yang terbentuk menggunakan ONNX dan NVIDIA *Tensor* RT API.

Tabel 3.3

FPS Model YOLOv7 *Mask* pada *No* GPU *Compile* dan GPU *Compiled*

RESOLUTION	***NO* GPU *COMPILE* (FPS)**	**GPU *COMPILED* (FPS)**
960p	15.35 FPS	42.45 FPS
640p	20.12 FPS	57.12 FPS
320p	40.25 FPS	134.24 FPS

Seperti yang dilihat pada Tabel 3.3, dengan melakukan optimasi dan *compile* pada model YOLOv7 *Mask* akan meningkatkan kecepatan *inference* model secara signifikan sehingga pada skripsi ini, optimisasi dan *compile* model YOLOv7 *Mask* akan diterapkan. Pendeteksian dan *mask* akan menentukan jika model rekognisi akan dijalankan, ketika model *mask* tidak mendeteksi satupun manusia pada suatu *frame*, maka *frame* data pergerakan kosong (*frame* data pergerakan yang tidak memiliki pergerakan) akan dimasukkan ke dalam *queue* dan model rekognisi tidak dijalankan pada *frame* tersebut.

3.2.2.4 Penyimpanan Hasil *Preprocessing* untuk *Training*

Meskipun proses *preprocessing* didesain untuk berjalan secepat mungkin, proses ini tetap saja memakan waktu dan akan memperlama tahap pengujian, *tuning* dan *training* model rekognisi karena proses ini akan diulang-ulang. Untuk mengurangi pengulangan *preprocessing*, diperlukan sistem penyimpanan hasil *preprocesing* agar video yang melalui tahap *preprocessing* tidak perlu dilakukan *preprocessing* lagi. Data-data hasil *preprocessing* seperti *motion vector*, *optical flow*, *masking*, dan *bounding box* berserta atribut-atribut seperti resolusi, jumlah *frames*, rekognisi, dan *path* video sumber disimpan pada satu file terstruktur HDF5 (*Hierarchical Data Format* v5). *Frame* data pergerakan seperti *motion vector* atau *optical flow* akan disimpan dalam bentuk aslinya (*unbounded*) sehingga tidak hilang karena proses normalisasi. Detail struktur data penyimpanan dijelaskan pada Tabel 3.4.

Tabel 3.4

Detail Struktur Data *File Preprocessing*

NAME	***TYPE***	***DETAILS***
Attributes		
*action*_id	uint8	Nomor identitas kategori aktifitas dari video yang di-*preprocess* (detail pada Tabel 3.2).
camera_id	uint8	Nomor identitas kamera dari video yang di-*preprocess.*
performer_id	uint8	Nomor identitas *performer* dari video yang di-*preprocess.*
replication_id	uint8	Nomor identitas replikasi dari video yang di-*preprocess.*
setup_id	uint8	Nomor identitas *setup* dari video yang di-*preprocess.*
generated_file_name	*string*	*Filename* dari *file preprocessing.*
source_file_path	*string*	Alamat sumber video yang di-*preprocess.*
mask_count	uint16	Jumlah total *frame masking.*
*mask*_hw	numpy.array[uint16]	Ukuran *frame masking* H (lebar *frame*) x W (panjang *frame*).
motion_vectors_are_raw	bool	*True* jika *frame motion vector* yang disimpan *unbounded*.
motion_vectors_bound	int16	*Limit* bound dari *frame motion vector.* Jika *motion_vectors_are_raw True*, maka nilai *motion_vectors_bound* adalah -1.
motion_vectors_count	uint16	Jumlah total *frame motion vector.*
*motion_vectors*_hwc	numpy.array[uint16]	Ukuran *frame motion vector* H (lebar *frame*) x W (panjang *frame*) x C (*channel /* X-*Axis* & Y-*Axis*).
optical_flows_are_raw	bool	*True* jika *frame optical flow* yang disimpan *unbounded*.

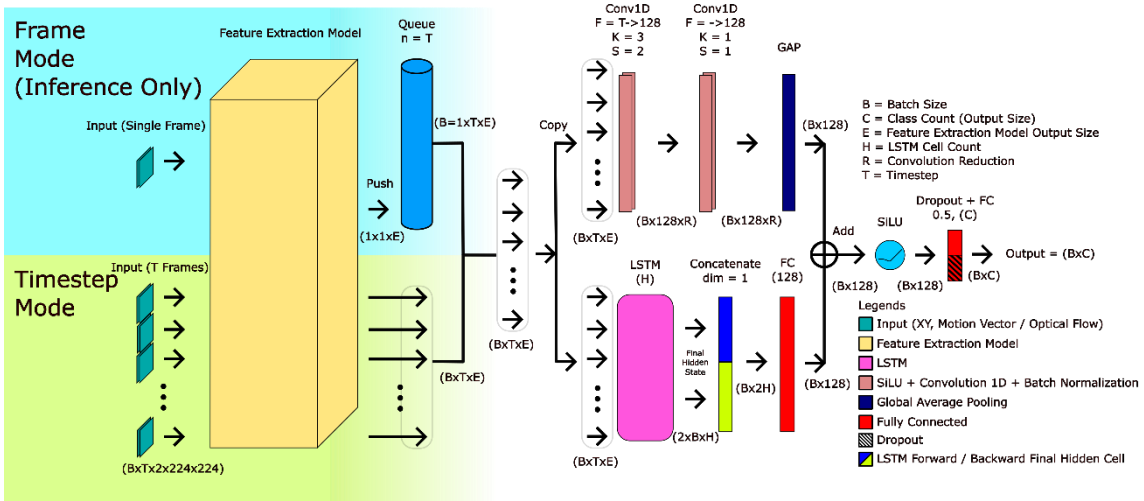
NAME	***TYPE***	***DETAILS***
optical_flows_bound	int16	*Limit* bound dari *frame optical flow.* Jika *optical_flows_are_raw True*, maka nilai *optical_flows_bound* adalah -1.
optical_flows_count	uint16	Jumlah total *frame optical flow.*
*optical_flows_*hwc	numpy.array[uint16]	Ukuran *frame optical flow* H (lebar *frame*) x W (panjang *frame*) x C (*channel /* X-*Axis* & Y-*Axis*).
optical_flows_ overlap_grid_mode	bool	*True* jika *frame optical flow* menggunakan *grid sampling*.
optical_flows_ overlap_grid_scale	int16	*Jumlah grid* yang digunakan pada *grid sampling* dari *frame optical flow.* Jika *optical_flows_overlap_grid_mode False*, maka nilai *optical_flows_overlap_grid_scale* adalah -1.
Datasets		
masks	numpy.array[bool]	*Array frame masking* hasil dari YOLOv7 - *Mask.* Memiliki ukuran N (jumlah *frame*) x H (lebar *frame*) x W (panjang *frame*).
masks_exist	numpy.array[bool]	*Array* yang menentukan jika ada manusia terdeteksi atau tidak per *frame*-nya.
masks_mcbb	numpy.array[int16]	*Array most center bounding box* hasil dari YOLOv7 – *Mask*. Memiliki ukuran N (jumlah *frame*) x 4 (X kiri, Y atas, X kanan, Y bawah).
motion_vectors	numpy.array[int16/uint16]	*Array frame motion vector.* Memiliki ukuran N (jumlah *frame*) x H (lebar *frame*) x W (panjang *frame*) x C (*channel /* X-*Axis* & Y-*Axis*). Jika *motion_vectors_are_raw True* maka tipe data adalah int16, jika *False* maka tipe data adalah uint16.

NAME	*TYPE*	*DETAILS*
optical_flows	numpy.array[int16/uint16]	*Array frame optical flow.* Memiliki ukuran N (jumlah *frame*) x H (lebar *frame*) x W (panjang *frame*) x C (*channel* / X-*Axis* & Y-*Axis*). Jika *optical_flows_are_raw True* maka tipe data adalah int16, jika *False* maka tipe data adalah uint16.

Untuk membaca data *preprocessing*, *mock class* dan *mock function* akan dibuat yang merupakan *inheritance* dari *class* dan *function* aslinya untuk menyimulasikan *output* dari fungsi asli agar *training* dan *testing* tidak perlu merubah kode banyak melainkan hanya merubah *class* yang dipakai (*training* memakai *mock class*, *testing* / *evaluation* menggunakan *class* asli).

3.2.3 Model Rekognisi Aktivitas Manusia

Model yang akan digunakan untuk melakukan rekognisi merupakan model yang menggunakan gabungan dari *Convolution Neural Network* (CNN) dan *Long Short-Term Memory* (LSTM) *Network*. Input dari model ini merupakan hasil *preprocessing* yang berupa *frame* data pergerakan yang berasal dari *motion vector* maupun *optical flow*. *Frame* data pergerakan yang terdiri dari dua *frame* yaitu horizontal (X-*Axis*) dan vertikal (Y-*Axis*) akan ditumpuk pada dimensi channelnya.

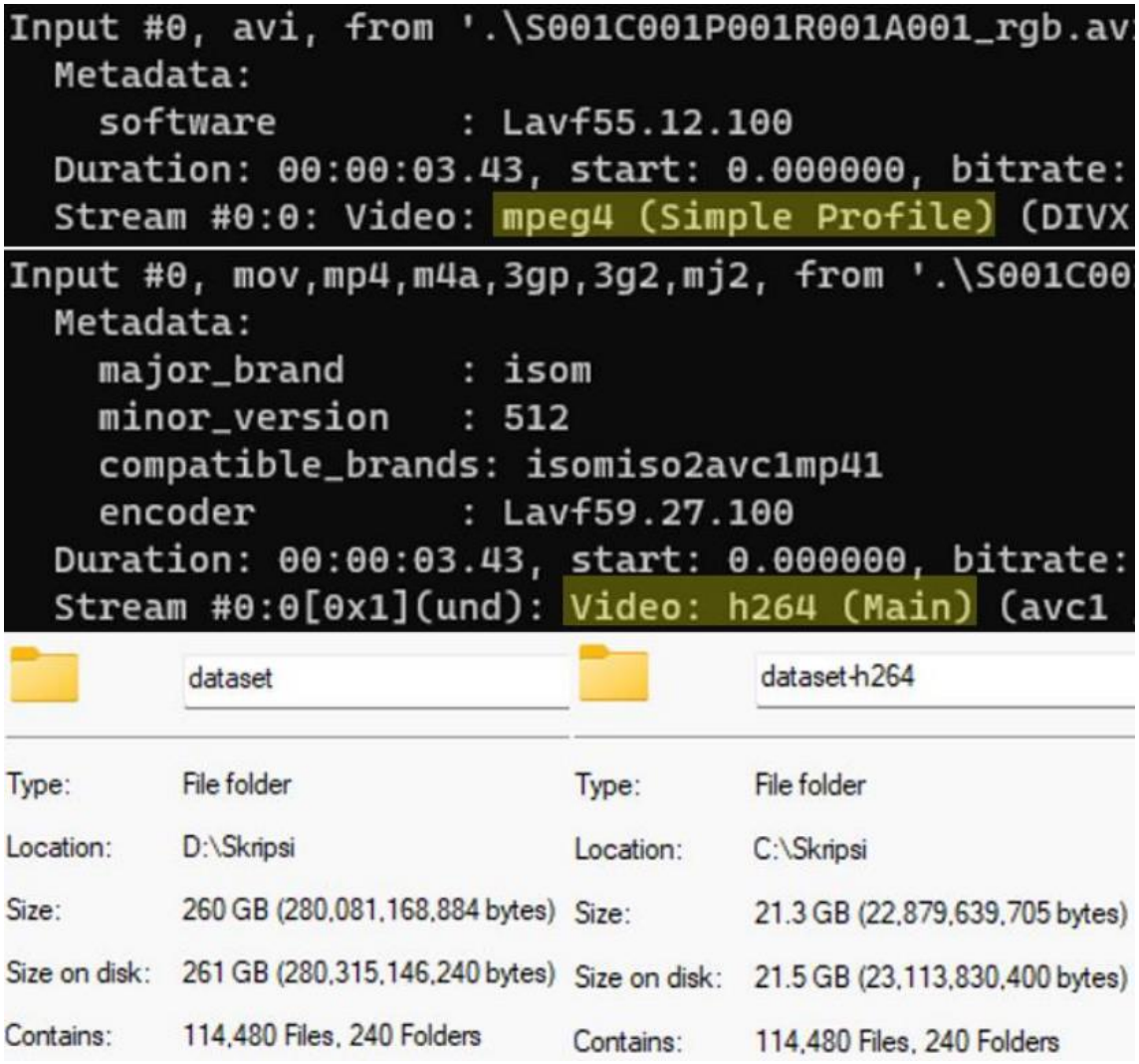


Gambar 3.14 *Ilustrasi* Model CNN-LSTM Rekognisi Aktivitas Manusia

Model yang diilustrasikan pada Gambar 3.14 *Ilustrasi* Model CNN-LSTM Rekognisi Aktivitas Manusia akan memiliki 2 jenis mode input, *timestep mode* membutuhkan input berukuran *batch x timesteps x 2 x 224 x 224* digunakan untuk melakukan *training* dan *batch inference*, model ini tidak menggunakan queue. *Mode* kedua yaitu *frame mode* hanya bisa digunakan pada *inference* menerima input berukuran 1 x 1 x 2 x 224 x 224 untuk melakukan *inference* bersama dengan *input* sebelum – sebelumnya, pada *mode* ini digunakan *queue* untuk menyimpan sejumlah *timesteps* banyaknya *output* dari model *feature extraction*. Setelah melalui model *feature extraction* untuk mendapatkan *feature – feature*, *feature – feature* ini akan menjadi input dari layer CNN dan layer LSTM. Kedua layer - layer CNN dan LSTM akan dijumlah pada akhirnya lalu hasil penjumlahan ini akan lanjut ke layer *fully connected* untuk menghasilkan *klasifikasi*.

3.2.4 Konversi *Encoding* Dataset

Video RGB pada dataset ini memiliki encoding MPEG-4 (bukan MPEG-4 Part 10 atau H.264). MPEG-4 memiliki *motion vector* tetapi implementasi *motion vector* pada *encoding* tersebut lebih primitif dari pada H.264 (tidak ada *Bi-Directional Frames*) dan MPEG-4 bukan *encoding* yang secara umum dipakai pada kamera CCTV dan media video pada umumnya, oleh karena itu video-video dari dataset akan dikonversi ke H.264 menggunakan aplikasi konversi media *open source ffmpeg*. Konversi ini juga dapat membantu mengurangi ukuran dari file video-video dataset secara signifikan tanpa kehilangan kualitas. Gambar 3.15 menunjukan pengurangan signifikan ukuran file video-video hingga 92 persen.



Gambar 3.15 Konversi dan Penghematan Storage Space dari MPEG-4 (Atas dan Kiri) ke H.264 (Bawah dan Kanan)