# DEPTHWISE SEPARABLE CONVOLUTION UNTUK REDUKSI PARAMETER PADA IDENTIFIKASI TINGKAT KESADARAN PENGEMUDI DALAM BERKENDARA DARI DATA VIDEO DENGAN CONVOLUTIONAL LSTM

**SKRIPSI**


**Oleh:**

**FAJAR FATHA ROMADHAN**

**200411100047**

**Dosen Pembimbing 1 : Dr. Indah Agustien Siradjuddin, S.Kom., M.Kom**

**Doesn Pembimbing 2 : Prof. Dr. Arif Muntasa, S.Si., M.T.**

**PROGRAM STUDI TEKNIK INFORMATIKA**

**JURUSAN TEKNIK INFORMATIKA**

**FAKULTAS TEKNIK**

**UNIVERSITAS TRUNOJOYO MADURA**

**2024**

**HALAMAN JUDUL**

# *DEPTHWISE SEPARABLE CONVOLUTION* UNTUK REDUKSI PARAMETER PADA IDENTIFIKASI TINGKAT KESADARAN PENGEMUDI DALAM BERKENDARA DARI DATA VIDEO DENGAN *CONVOLUTIONAL* LSTM

**SKRIPSI**

Diajukan untuk Memenuhi Persyaratan Penyelesaian Studi Strata Satu (S1) dan Memperoleh Gelar Sarjana Komputer (S. Kom.) di Universitas Trunojoyo Madura


**Disusun Oleh:**

**Fajar Fatha Romadhan**

**200411100047**

**PROGRAM STUDI TEKNIK INFORMATIKA**

**JURUSAN TEKNIK INFORMATIKA**

**FAKULTAS TEKNIK**

**UNIVERSITAS TRUNOJOYO MADURA**


**2024**

**HALAMAN PENGESAHAAN**

**_DEPTHWISE SEPARABLE CONVOLUTION_ UNTUK REDUKSI PARAMETER PADA IDENTIFIKASI TINGKAT KESADARAN PENGEMUDI DALAM BERKENDARA DARI DATA VIDEO DENGAN _CONVOLUTIONAL_ LSTM**

Skripsi disusun untuk memenuhi syarat memperoleh gelar Sarjana Komputer (S. Kom.) di Universitas Trunojoyo Madura

**Oleh:**

**Nama: Fajar Fatha Romadhan**
**NIM : 200411100047**

**Disetujui oleh Tim Penguji Skripsi:** **Tanggal Sidang: 17 Juli 2024**

**Dr. Indah Agustien Siradjuddin, S.Kom., M.Kom.** (Pembimbing I)
**NIP. 197808202002122001**

**Prof. Dr. Arif Muntasa, S.Si., M.T.** (Pembimbing II)
**NIP. 196911182001121004**

**Dr. Noor Ifada, S.T., MISD.** (Penguji I)
**NIP. 197803172003122001**

**Dr. Arik Kurniawati, S.Kom., M.T.** (Penguji II)
**NIP. 197803092003122009**

**Devie Rosa Anamisa, S.Kom., M.Kom.** (Penguji III)
**NIP. 198411042008122003**

**Mengetahui,**

**Ketua Jurusan Teknik Informatika**

**Dr. Yeni Kustiyahningsih, S.Kom., M.Kom.**
**NIP. 197709212008122002**

## HALAMAN PERNYATAAN ORISINALITAS

Saya yang bertanda tangan dibawah ini, menyatakan bahwwa skripsi dengan judul:

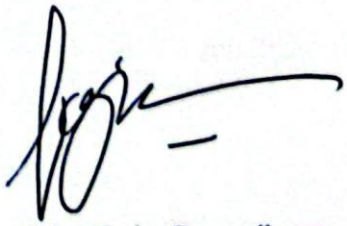
"*DEPTHWISE SEPARABLE CONVOLUTION* UNTUK REDUKSI PARAMETER PADA IDENTIFIKASI TINGKAT KESADARAN PENGEMUDI DALAM BERKENDARA DARI DATA VIDEO DENGAN *CONVOLUTIONAL* LSTM"

1. Adalah asli, bukan merupakan karya pihak lain serta belum pernah diajukan untuk mendapatkan gelar akademik Sarjana Komputer baik di Universitas Trunojoyo Madura maupun di Perguruan Tinggi yang lain di Indonesia.
2. Tidak terdapat karya atau pendapat pihak lain yang pernah ditulis atau diterbitkan oleh orang lain, kecuali yang secara tertulis telah diacu dalam naskah ini dan disebutkan dalam daftar pustaka.

Apabila dikemudian hari terbukti skripsi ini sebagian atau seluruhnya merupakan hasil plagiasi atau terdapat hal-hal yang tidak sesuai dengan pernyataan di atas, maka saya sanggup menerima sanksi akademis yang berlaku dengan segala akibat hukumnya sesuai peraturan Universitas Trunojoyo Madura dan atau peraturan perundang-undangan yang berlaku.

Bangkalan, 29 Juli 2024

Yang menyatakan



Fajar Fatha Romadhan

200411100047

# KATA PENGANTAR

Alhamdulillah puji syukur penulis panjatkan kepada Allah Subhanahu wa ta'ala atas rahmat serta hidayah-Nya penulis dapat menyelesaikan Skripsi yang berjudul "*Depthwise Separable Convolution* untuk Reduksi Parameter Pada Identifikasi Tingkat Kesadaran Pengemudi dalam Berkendara dari Data Video dengan *Convolutional* LSTM". Dari beberapa penelitian sebelumnya tentang identifikasi kesadaran pengemudi dan sejenisnya, metode ini telah terbukti efektif dalam mengenali permasalahan dengan akurasi yang baik. Meskipun dalam percobaan yang peneliti lakukan menghasilkan hasil yang kurang baik.

Dengan terselesaikannya laporan skripsi ini, penulis menyampaikan terima kasih kepada:

1. Orang tua yang selalu memberikan dukungan baik moral maupun materi.
2. Dr. Indah Agustien Siradjuddin, S.Kom., M.Kom. selaku Dosen Pembimbing I dan Prof. Dr. Arif Muntasa, S.Si., M.T. selaku Dosen Pembimbing II.
3. Bapak dan Ibu Dosen Fakultas Teknik, khususnya program studi Teknik Informatika Universitas Trunojoyo Madura.
4. Pihak lain yang tidak bisa disebutkan satu-persatu.

Penulis menyadari bahwa skripsi ini tidak terlepas dari berbagai kesalahan. Oleh karena itu, penulis menyatakan permohonan maaf atas segala kesalahan dan kekurangan yang ada. Dengan kerendahan hati, berbagai saran dan kritik yang membangun sangat penulis harapkan dari rekan-rekan pembaca.

Bangkalan, 26 Juli 2024

Penulis

# ABSTRAK

Salah satu faktor utama penyebab kecelakaan lalu lintas yaitu mengemudi dengan kondisi mengantuk. Model yang digunakan ntuk mengidentifikasi kantuk pengemudi umumnya hanya akan mengenali pola dari data tanpa memperhatikan urutan waktu pada data berurutan atau *time series*. Metode yang digunakan pada penelitian ini yaitu *Convolutional Long Short Term Memory* atau *Convolutional* LSTM. *Convolutional* LSTM dinilai efektif untuk mengolah data *spatio temporal*. Data yang digunakan merupakan data video dari seseorang yang sedang mengemudi. Pada penelitian ini *Depthwise Separable Convolution* digunakan untuk mereduksi jumlah parameter pada operasi konvolusi pada setiap *gate Convolutional* LSTM. Penelitian ini bertujuan untuk mengetahui apakah metode *Convolutional* LSTM yang dikombinasikan dengan *Depthwise Separable Convolution* dapat dengan efektif dan efisien untuk mendeteksi kantuk pengemudi dari data video. Dari penelitian yang telah dilakukan dengan penurunan parameter sejumlah 12.23%, proses pelatihan secara keseluruhan mengalami penurunan waktu sebesar 33.93%. Pada proses evaluasi, rata-rata dari keseluruhan f1-*score* mengalami penurunan sebesar 1.60%. Selain itu, rata-rata dari keseluruhan waktu prediksi pada saat evaluasi mengalami penurunan sebesar 33.21%. Dari hasil yang didapatkan disimpulkan bahwa *Convolutional* LSTM dapat dengan cukup baik mengenali kantuk pengemudi dari data video. Sedangkan implementasi *Depthwise Separable Convolution* dapat secara efektif mengurangi jumlah parameter pada metode *Convolutional* LSTM yang mengakibatkan penurunan waktu pelatihan dan prediksi tanpa mengurangi performanya secara signifikan.



**Kata kunci:** *Convolutional*, Data Video, *Depthwise Separable*, Kantuk Pengemudi, *Long Short Term Memory*

## ABSTRACT

*One of the main factors causing traffic accidents is driving while drowsy. The model used to identify driver drowsiness generally only recognizes patterns from data without considering the time sequence in sequential data or time series. The method used in this study is Convolutional Long Short Term Memory or Convolutional LSTM. Convolutional LSTM is considered effective for processing spatiotemporal data. The data used is video data from someone driving. In this study, Depthwise Separable Convolution is used to reduce the number of parameters in the convolution operation on each Convolutional LSTM gate. This study aims to determine whether the Convolutional LSTM method combined with Depthwise Separable Convolution can be effective and efficient in detecting driver drowsiness from video data. From the research that has been done with a decrease in parameters of 12.23%, the overall training process has decreased by 33.93%. In the evaluation process, the average of the overall f1-score decreased by 1.60%. In addition, the average of the overall prediction time during the evaluation decreased by 33.21%. From the results obtained, it is concluded that Convolutional LSTM can quite well recognize driver drowsiness from video data. Meanwhile, the implementation of Depthwise Separable Convolution can effectively reduce the number of parameters in the Convolutional LSTM method which results in a decrease in training and prediction time without significantly reducing its performance.*



***Keywords**: Convolutional, Video Data, Depthwise Separable, Driver Drowsiness, Long Short Term Memory*

# DAFTAR ISI

# DAFTAR GAMBAR

# DAFTAR TABEL

## DAFTAR LAMPIRAN

# BAB I
# PENDAHULUAN

## 1.1. Latar Belakang

Kecelakaan lalu lintas dapat terjadi karena banyak faktor. Salah satu faktor utama penyebab kecelakaan lalu lintas yaitu mengemudi dengan kondisi mengantuk.. Menurut data dari *World Health Organization* (WHO), 20% kecelakaan lalu lintas di dunia disebabkan karena pengemudi yang mengantuk. Berdasarkan data dari *National Highway Traffic Safety Administration* (NHTSA) di Amerika Serikat, setiap tahunnya lebih dari 1,25 juta kematian dan 20 sampai 50 juta orang terluka bahkan cacat yang disebabkan oleh kecelakaan yang terjadi karena mengemudi dengan kondisi mengantuk [1]. Di indonesia, angka kecelakaan lalu lintas yang disebabkan pengemudi yang mengantuk tiap tahunnya juga termasuk sangat tinggi. Karena tingginya angka kecelakaan yang disebabkan oleh pengemudi yang mengantuk tersebut, maka diperlukan tindakan yang cepat dan tepat [2].

Seseorang umumnya tidak akan menyadari jika pengemudi dalam kondisi mengantuk. Untuk mengidentifikasi kantuk pengemudi dapat dilakukan dengan menerapkan berbagai cara, salah satunya dengan memanfaatkan teknologi komputer. Dengan memanfaatkan teknologi komputer, pengemudi yang mengantuk dapat diketahui melalui sistem sehingga dapat dilakukan upaya antisipasi terjadinya kecelakaan. Pengenalan cepat terhadap pengemudi yang mengantuk mungkin akan mengurangi jumlah kecelakaan lalu lintas [3].

Model pembelajaran mesin biasa digunakan untuk mengidentifikasi kantuk pengemudi. Pembelajaran mesin merupakan bagian dari kecerdasan buatan (*Artificial Intelligence*) yang mengimplikasikan program-program untuk memicu komputer sehingga dapat berperilaku cerdas seperti pada manusia. Selain itu, kecerdasan buatan dapat mengembangkan pemahaman mereka melalui pengalaman yang mereka peroleh secara otomatis melalui sebuah pelatihan [4]. Meskipun demikian, model pembelajaran mesin umumnya hanya mempelajari pola dari data tanpa mempertimbangkan atau memperhatikan urutan waktu pada data berurutan atau *time series*.

Beberapa tahun terakhir penelitian tentang identifikasi tingkat kesadaran pengemudi yang berupa kantuk sudah banyak dilakukan. Seperti pada penelitian tahun 2019 oleh You, *et. al.* [5]. Penelitian ini membahas tentang deteksi kantuk berkendara secara real-time dengan mempertimbangkan perbedaan individu. Penelitian ini dilakukan dengan beberapa tahapan yaitu ekstraksi *frame,* deteksi wajah menggunakan *Deep Cascaded Convolutional Neural Network* atau DCCNN, dan pemberian *landmarks* wajah. Dari *landmarks* wajah didapatkan *Eyes Aspect Ratio* atau EAR. Dari EAR inilah dilakukan proses pelatihan menggunakan *Support Vector Machine* atau SVM. Dari proses pelatihan dan evaluasi yang dilakukan didapatkan akurasi 94,80%.

Selain itu terdapat juga penelitian pada tahun 2021 oleh Altameem, *et. al.* [6]. Penelitian ini tentang identifikasi dan deteksi dini kantuk pengemudi dengan *hybrid machine learning*. Penelitian ini dilakukan melalui beberapa pemrosesan yaitu, ekstraksi *frame,* deteksi wajah, lalu dilakukan monitoring mata. Proses pelatihan dilakukan dengan menggunakan SVM. Proses pengujian atau evaluasi dilakukan pada beberapa skenario. Skenario pertama pada saat siang hari dengan kondisi normal memiliki akurasi 91%, skenario kedua pada siang hari dengan pencahayaan redup memiliki akurasi 81%, skenario ketiga pada malam hari dengan pencahayaan terang menghasilkan akurasi 93%, dan skenario 4 pada malam hari dengan pencahayaan redup menghasilkan akurasi 68%.

Dari beberapa penelitian terkait, metode yang digunakan bukan merupakan metode yang secara khusus diterapkan untuk menangani data berurutan atau *time series.* Metode pembelajaran mesin yang dapat digunakan untuk memproses dan mengolah data berurutan atau *time series* salah satunya yaitu *Recurrent Neural Network* (RNN). Dengan memanfaatkan konsep memori *internal* atau *state,* RNN memungkinkan untuk mengingat dan mengolah informasi *input* dari *time step* sebelumnya sehingga dapat mempengaruhi pemrosesan informasi saat ini. RNN memiliki kemampuan yang baik untuk mengeksploitasi ketergantungan temporal jangka panjang dan korelasi variabel [7].

RNN secara teori dapat mempelajari ketergantungan jangka panjang yang jauh dari masa lampau. Hal tersebut mengakibatkan semua *input* sebelum-

sebelumnya dapat memengaruhi nilai *output*. Akan tetapi pada penerapannya RNN memiliki masalah yang terkait dengan korelasi waktu dari *input*, yaitu *input* yang diproses pada RNN terlalu jauh dari masa lampau. Untuk mengatasi masalah tersebut RNN dikembangkan lagi menjadi sebuah metode yang dinamakan *Long Short Term Memory* (LSTM). LSTM merupakan pengembangan dari RNN yang dinilai dapat mengatasi masalah pada korelasi urutan waktu baik dalam waktu yang singkat maupun lama [8] dengan memanfaatkan lapisan tersembunyi (*hidden layer/hidden state*) sebagai sel memori. Jaringan LSTM dapat digunakan untuk mempelajari karakteristik, mengidentifikasi, dan mengenali peristiwa secara otomatis. LSTM terbukti efektif untuk mempelajari ketergantungan jarak jauh dalam berbagai penelitian sebelumnya [9].

Pada Implementasinya LSTM tidak dapat menerima dan mengolah *input* yang berupa data spatial seperti pada citra secara langsung. *Input* yang berupa citra sebelumnya harus dilakukan ekstraksi fitur dan diubah dimensinya sedemikian rupa agar dapat diproses atau diolah oleh LSTM. Hal tersebut mengakibatkan data yang diproses pada LSTM sudah tidak mempertahankan fitur spatialnya. Data video merupakan data berurutan yang terdiri dari sejumlah *frame* yang berbentuk citra. Pada data video setiap frame memiliki ketergantungan untuk menentukan frame berikutnya. Oleh karena itu data video merupakan data yang mengandung fitur *spatial* dan *temporal*. Untuk memproses data video dengan mempertahankan fitur spatialnya, LSTM dapat memanfaatkan operasi konvolusi yang diterapkan pada arsitekturnya. Arsitektur ini disebut *Convolutional* LSTM [10]. Dalam Convolutional LSTM, operasi konvolusi berguna untuk mengolah fitur spatial dari citra agar dapat diproses langsung oleh LSTM sehingga *Convolutional* LSTM dapat mengolah dan mempelajari urutan dari setiap frame pada video. Convolutional LSTM dinilai dapat menangkap informasi *temporal* dan *spatial* dan memberikan kinerja superior pada frame video dibandingkan dengan *fully connected* LSTM [11][12]. Terdapat banyak penelitian yang telah dilakukan menggunakan metode *Convolutional* LSTM untuk data video seperti pada penelitian Zhang et. al. [13] tahun 2018, Ye et. al. [14] tahun 2019, Duman dan Erdem [11] tahun 2019, dan Mobsite et. al. [15] tahun 2023. Penelitian tersebut menghasilkan akurasi yang cukup baik.

Operasi konvolusi pada citra merupakan sebuah operasi yang berguna untuk mendapatkan dan mengolah fitur spatial pada citra. Konvolusi bekerja dengan mengalikan kernel atau filter konvolusi dengan citra *input* dan digeser secara berurutan hingga melintasi seluruh citra *input* tersebut. Hasil dari perkalian kernel dengan citra *input* tersebut dijumlahkan sehingga menghasilkan nilai tunggal yang nantinya menghasilkan *feature map*. Pada proses konvolusi biasanya memiliki jumlah *trainable parameter* yang cukup besar. Jumlah parameter pada konvolusi dipengaruhi oleh banyaknya kernel, ukuran kernel, dan channel citra *input* [16].

*Depthwise Separable Convolution* merupakan salah satu metode konvolusi yang dapat mereduksi atau mengurangi jumlah parameter pada proses konvolusi. *Depthwise Separable Convolution* merupakan jenis konvolusi yang dinilai efektif mengurangi jumlah parameter dari proses konvolusi biasa [17]. Dengan jumlah parameter dan komputasi yang lebih sedikit, arsitektur model dapat melakukan proses pelatihan yang lebih cepat tanpa mengurangi kemampuannya secara signifikan [18].

Penelitian ini bertujuan untuk mengembangkan sebuah model yang diharapkan mampu mengidentifikasi kantuk pada pengemudi dengan cara mengenali dan mempelajari pola perubahan wajah pada pengemudi seiring dengan waktu dengan memanfaatkan metode yang secara khusus dapat menangani data berurutan atau *time series* yang berupa video. Metode yang digunakan adalah *Convolutional LSTM* yang dikombinasikan dengan *Depthwise Separable Convolution* atau dapat disebut juga *Depthwise Separable Convolutional* LSTM. Metode *Convolutional* LSTM dianggap dapat mempelajari fitur *spatio-temporal* dari data video dengan baik. Sedangkan *Depthwise Separable Convolution* dinilai dapat mereduksi jumlah parameter pada proses konvolusi. Dengan jumlah parameter yang direduksi diharapkan model dapat lebih efisien tanpa mengurangi performanya secara signifikan [18].

## 1.2. Rumusan Masalah

### 1.2.1. Permasalahan

Untuk mengidentifikasi kantuk pengemudi dapat dilakukan dengan mengenali dan mempelajari pola perubahan wajah pada pengemudi seiring dengan

waktu. Pada penelitian sebelumnya yang menggunakan data video, metode yang digunakan untuk membuat model identifikasi kantuk bukan merupakan metode yang secara khusus diterapkan untuk menangani data berurutan atau *time series*, sehingga model hanya mempelajari pola dari data tanpa mengenali urutan waktu. Sedangkan data video merupakan data yang memiliki fitur *spatial* dan *temporal*. Selain itu masalah jumlah parameter yang tinggi sering kali menyebabkan waktu pelatihan dan prediksi yang relatif lebih lama [18].

### 1.2.2. Solusi Permasalahan

Solusi yang diusulkan dalam penelitian ini yaitu, pembuatan model dilakukan dengan mengenali dan mempelajari pola pergerakan atau perubahan wajah pada pengemudi seiring dengan waktu dengan menerapkan metode atau algoritma *Convolutional Long Short Term Memory* atau *Convolutional* LSTM. *Convolutional* LSTM merupakan arsitektur LSTM yang dikombinasikan dengan proses konvolusi. *Convolutional* LSTM dinilai dapat menerima dan mengolah *input* spatial seperti citra dan menyimpan informasi spatialnya. Operasi konvolusi berguna untuk mengolah fitur spatial dari data sehingga dapat diteruskan pada LSTM untuk mempelajari fitur tersebut berdasarkan urutan waktu. Untuk membuat metode lebih efisien digunakan *Depthwise Separable Convolution* yang menggantikan operasi konvolusi pada setiap *gate Convolutional LSTM*. Implementasi *Depthwise Separable Convolution* pada Convolutional LSTM dinilai dapat mereduksi atau mengurangi jumlah parameter dikarenakan *Depthwise Separable Convolution* memiliki jumlah parameter yang lebih sedikit daripada operasi konvolusi pada umumnya.

### 1.2.3. Pertanyaan Penelitian

Dari permasalahan yang telah di jelaskan di atas, dapat dirumuskan beberapa pertanyaan penelitian sebagai berikut:

1. Berapa nilai F1-Score dari model yang dihasilkan menggunakan metode *Convolutional* LSTM yang dikombinasikan dengan *Depthwise Separable Convolution* dalam klasifikasi video kantuk pengemudi?
2. Bagaimana pengaruh implementasi *Depthwise Separable Convolution* terhadap waktu komputasi dan nilai F1-*Score*?

## 1.3. Tujuan dan Manfaat

Dari pertanyaan penelitian yang dirumuskan dapat diperoleh tujuan dan manfaat dari penelitian yang dilakukan.

### 1.3.1. Tujuan

Berdasarkan pertanyaan penelitian, tujuan dari penelitian ini dirumuskan dalam 2 poin sebagai berikut:

1. Untuk mengetahui nilai F1-Score dari model yang dihasilkan menggunakan metode *Convolutional* LSTM yang dikombinasikan dengan *Depthwise Separable Convolution* dalam klasifikasi video kantuk pengemudi.
2. Untuk mengetahui pengaruh implementasi *Depthwise Separable Convolution* terhadap waktu komputasi dan nilai F1-*Score.*

### 1.3.2. Manfaat

Dari penelitian ini diharapkan mampu mengembangkan model yang efektif dan efisien untuk mengidentifikasi kantuk pengemudi dengan menggunakan data video. Sehingga dapat dikembangkan menjadi sebuah sistem yang dapat mengenali ketika pengemudi mengantuk dan memberikan peringatan. Hasil dari penelitian ini diharapkan dapat memberikan manfaat pada bidang akademik serta dapat memberikan wawasan dari penerapan *Convolutional* LSTM dan *Depthwise Separable Convolution* untuk data *time series* yang berupa video. Selain itu, diharapkan juga dapat mendukung solusi baru dalam bidang *sequence processing* dan dapat digunakan sebagai rujukan untuk penelitian berikutnya.

## 1.4. Batasan Masalah

Batasan masalah yang ditetapkan dalam penelitian ini antara lain:

1. Dataset yang digunakan yaitu SUST-DDD atau Sivas University of Science and Technology Driver Drowsiness Dataset (2022).
2. Jumlah kelas yang diklasifikasikan adalah 2, yaitu mengantuk dan tidak mengantuk.
3. Jumlah data yang digunakan yaitu 2074 dengan rincian kelas mengantuk 975 data tidak mengantuk 1099 data.
4. Dataset berupa video wajah dari seseorang yang sedang mengemudi.

5. Metode pengukuran yang digunakan untuk evaluasi model adalah menggunakan *Confusion Matrix* yang nantinya digunakan untuk menghitung F1-*score.*
6. Tujuan utama penelitian tidak untuk mencari model dengan efektifitas terbaik, tetapi lebih fokus pada pengaruh implementasi *Depthwise Separable Convolution* terhadap jumlah parameter dan akibatnya.
7. Fokus penelitian adalah untuk mengetahui apakah metode yang diusulkan efektif untuk permasalahan dan studi kasus yang dipilih. Tidak sampai pada implementasi model pada kasus nyata.

## 1.5. Sistematika Penulisan Skripsi

Pada penelitian ini sistematika penulisan skripsi yang digunakan adalah sebagai berikut:

**BAB I PENDAHULUAN**

Bab satu merupakan pendahuluan yang memiliki gambaran secara umum mengenai judul yang diangkat dalam penelitian ini, mulai dari pemaparan tingkat kecelakaan lalu lintas yang diakibatkan oleh pengemudi yang mengantuk sampai dengan *Convolutional* LSTM dan *Depthwise Separable Convolution*. Penulis menyusun pemaparan dan ringkasan dari setiap sub bab yang dibagi dalam lima sub bab yaitu latar belakang, perumusan masalah, tujuan dan manfaat penelitian, batasan masalah, dan sistematika penulisan.

**BAB II KAJIAN PUSTAKA**

Pada bab dua berisi tentang kajian penelitian terdahulu yang melandasi peneliti dalam menulis proposal ini, dan memuat kajian pustaka tentang landasan teori kantuk pengemudi, data *Time Series*, citra digital, konvolusi, *Depthwise Separable Convolution*, RNN, LSTM, *Convolutional* LSTM, dan Evaluasi Model.

**BAB III METODE USULAN**

Pada bab tiga menjelaskan tentang metode penelitian. Pada bab ini memuat jenis dan sumber dari data yang diambil, bab ini juga menguraikan teknik pengolahan data, langkah – langkah dalam penelitian dan evaluasi hasil analisa klasifikasi serta uji coba dalam proses analisa data dan uji coba yang akan diterapkan dalam penelitian.

**BAB IV HASIL DAN PEMBAHASAN**

Pada bab empat menjelaskan tentang penelitian yang dilakukan. Bab ini memuat kebutuhan lingkungan penelitian, pembuatan kode program, pelatihan, evaluasi, dan pembahasannya.

**BAB V PENUTUP**

Pada bab lima berisi tentang kesimpulan dan saran berdasarkan penelitian dan uji coba yang telah dilakukan.

# BAB II
# KAJIAN PUSTAKA

## 2.1. Kantuk Pengemudi

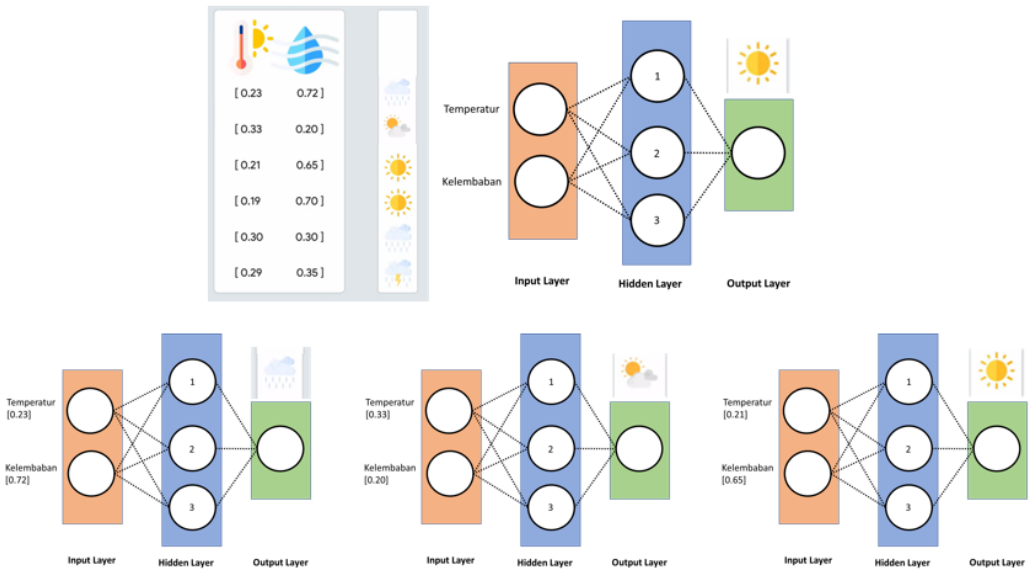
Kantuk adalah suatu kondisi dimana seseorang merasa ingin tidur dan tidak memiliki kemampuan untuk tetap terjaga atau sadar. Kantuk disebabkan oleh banyak faktor, antara lain kurang tidur, stres, kelelahan fisik atau mental, dan efek samping pengobatan tertentu. Rasa kantuk yang berlebihan, terutama pada saat melakukan pekerjaan yang membutuhkan konsentrasi tinggi, dapat mengganggu pekerjaan tersebut atau bahkan dapat mengancam keselamatan. Salah satu contohnya adalah pada saat mengemudi. Mengemudi sangat beresiko jika dilakukan dalam keadaan mengantuk [19].

Kecelakaan lalu lintas dapat terjadi karena banyak faktor. Salah satu faktor utama penyebab kecelakaan lalu lintas yaitu mengemudi dengan kondisi mengantuk.. Menurut data dari *World Health Organization* (WHO), 20% kecelakaan lalu lintas di dunia disebabkan karena pengemudi yang mengantuk. Berdasarkan data dari *National Highway Traffic Safety Administration* (NHTSA) di Amerika Serikat, setiap tahunnya lebih dari 1,25 juta kematian dan 20 sampai 50 juta orang terluka bahkan cacat yang disebabkan oleh kecelakaan yang terjadi karena mengemudi dengan kondisi mengantuk [1]. Di indonesia, angka kecelakaan lalu lintas yang disebabkan pengemudi yang mengantuk tiap tahunnya juga termasuk sangat tinggi. Tingginya angka kecelakaan disebabkan oleh berbagai faktor. Salah satunya adalah kantuk pengemudi [20].

## 2.2. Data *Time Series*

Data *time series* adalah sekumpulan data pencatatan hasil observasi yang didasarkan pada sifat kuantitatif dari satu atau lebih peristiwa yang terjadi sepanjang waktu.. Konsep time series data dapat dilihat pada Gambar 2. 1.

Pada Gambar 2. 1 merupakan konsep dari data time series yang diimplementasikan pada data cuaca dengan fitur temperatur dan kelembaban. Pada data ketiga atau hari ketiga terdapat output yaitu berawan, untuk memprediksi pada hari ketiga tersebut dibutuhkan data pada hari sebelumnya yaitu hari kedua dan kesatu. begitu juga untuk memprediksi pada hari ke empat yang membutuhkan data pada hari ketiga dan kedua




**Gambar 2. 1. Data Time Series**

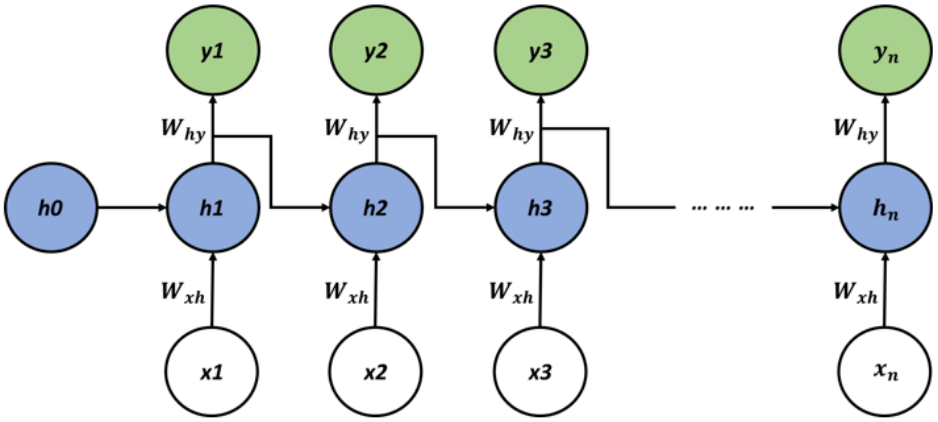
Jika konsep ini di implementasikan ke *Neural Network*, maka saat belajar atau memproses data pada hari ketiga tidak membutuhkan data sebelumnya. Karena *Neural Network* hanya mempelajari satu data per satu waktu. Maka dari itu dibutuhkan modifikasi dari arsitektur *Neural Network* ke arsitektur yang lebih baik yaitu salah satunya yaitu Recurrent Neural Network (RNN).

## 2.3. Recurrent Neural Network

Salah satu metode dalam *machine learning* yang dapat digunakan untuk memproses data sekuensial atau *time series* yaitu *Recurrent Neural Network* (RNN). Pada data berurutan atau *time series*, nilai pada data t+1 akan dipengaruhi oleh data t0, t-1, dan seterusnya. Dengan memanfaatkan konsep memori *internal* atau *state*, RNN memungkinkan untuk mengingat dan mengolah informasi *input* dari *time step* sebelumnya sehingga dapat mempengaruhi pemrosesan informasi saat ini. Pada setiap langkah, RNN menerima input dan memperbarui state internalnya dengan menggabungkan *input* saat ini dengan *state* sebelumnya. State ini kemudian digunakan untuk menghasilkan output dan diteruskan ke langkah berikutnya. Proses ini berulang untuk setiap langkah dalam *Dataset*. RNN mampu untuk mempertahankan informasi dengan menggunakan memori *internal* yang dapat digunakan untuk mengenali pola yang dinamis. Dengan menyimpan informasi sementara, RNN dapat menggunakannya dalam pengolahan data pada

langkah selanjutnya[21]. Beberapa karya terbaru mengungkapkan bahwa RNN lebih baik dalam memodelkan struktur temporal [22].

*Recurrent Neural Network* atau RNN memiliki arsitektur yang terdiri dari *input layer*, *hidden layer* dan *output layer*. Pada setiap koneksinya terdapat *weight*/bobot. *Input Layer* merupakan lapisan paling luar yang menghubungkan antara data yang dimasukkan ke dalam pemrosesan (*hidden layer*). *Hidden layer* merupakan lapisan yang digunakan untuk memproses variabel-variabel *input* agar dapat menghasilkan sebuah *output*. *Output layer* merupakan lapisan paling akhir yang digunakan untuk mengeluarkan hasil pemrosesan. Output yang didapatkan dipengaruhi oleh bobot, fungsi aktivasi dan, jumlah *hidden layer*. Gambar arsitektur RNN dapat dilihat pada Gambar 2. 2. **Arsitektur RNN**



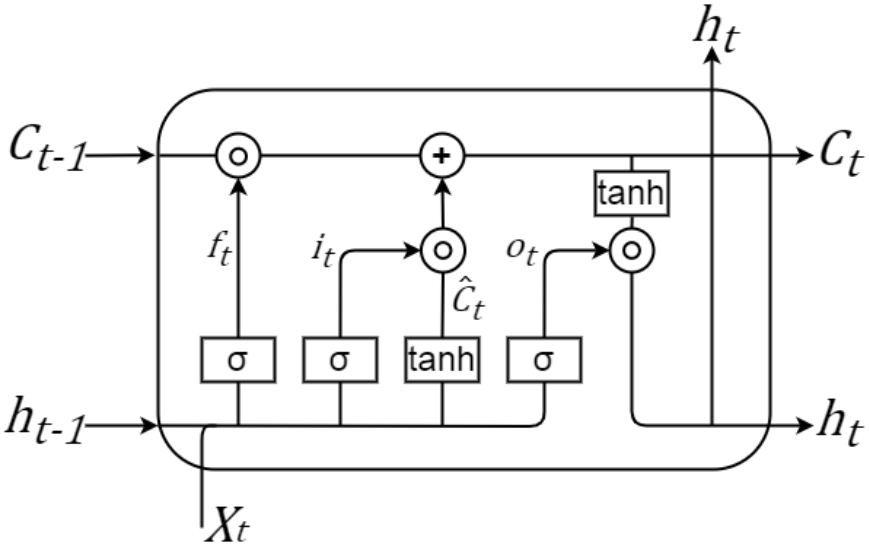

**Gambar 2. 2. Arsitektur RNN**

## 2.4. *Long Short Term Memory*

RNN secara teori dapat mempelajari ketergantungan jangka panjang yang jauh dari masa lampau. Hal tersebut mengakibatkan semua *input* sebelum-sebelumnya dapat memengaruhi nilai *output*. Akan tetapi pada penerapannya RNN memiliki masalah yang terkait dengan korelasi waktu dari *input*, yaitu *input* yang diproses pada RNN terlalu jauh dari masa lampau. Untuk mengatasi masalah tersebut RNN dikembangkan lagi menjadi sebuah metode yang dinamakan *Long Short Term Memory* (LSTM).

LSTM merupakan pengembangan dari RNN yang dinilai dapat mengatasi masalah pada korelasi urutan waktu baik dalam waktu yang singkat maupun lama [8] dengan memanfaatkan lapisan tersembunyi (*hidden layer/hidden state*) sebagai

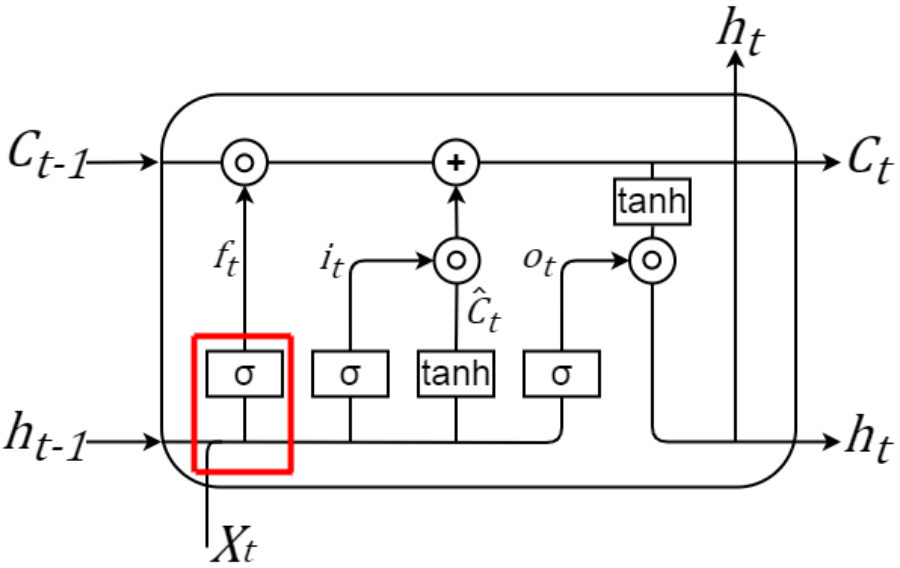
sel memori. Selain itu disebutkan juga bahwa LSTM adalah jenis RNN yang tidak menimbulkan masalah ketergantungan jangka panjang seperti yang dilakukan RNN sehingga menghindari masalah yang muncul pada gradien gradien yang lebih dalam dari jaringan [23]. LSTM memiliki keistimewaan selain terdapat *hidden state* yang akan dilanjutkan ke *time step* berikutnya terdapat juga *cell state*. *Cell state* ini dapat membawa informasi dari *time step* sebelumnya. Informasi tersebut nantinya dapat dibuang atau dapat juga digunakan pada *time step* berikutnya sehingga nantinya dapat diproses pada *hidden state*. Pada LSTM memiliki 3 gate utama yang terdiri dari *Forget Gate, Input Gate, dan Output Gate*. Arsitektur dari LSTM dapat diperhatikan pada Gambar 2. 3.




**Gambar 2. 3. Diagram LSTM**

Dalam diagram yang dipaparkan pada Gambar 2. 3 yang pertama dilakukan yaitu menghitung *forget gate*. Informasi dari *time step* sebelumnya yang akan dilupakan atau dilanjutkan pada proses pengolahan informasi berikutnya diatur pada *Forget gate*. Diagram LSTM untuk *forget gate* dapat diperhatikan pada Gambar 2. 4.

*Forget gate* didapatkan dari hasil *dot product input* pada *time step* saat ini dengan *hidden state* dari *time step* sebelumnya dengan *weight* dan *bias* yang selanjutnya dimasukkan ke dalam fungsi sigmoid. Untuk menghitung *Forget gate* dapat dilakukan dengan menggunakan Persamaan (2.1). dengan "•" merupakan *dot product* dan "○" hadamard product atau perkalian antar elemen.




**Gambar 2. 4. Forget Gate**

$$f_t = \sigma(W_{hf} \bullet h_{t-1} + W_{xf} \bullet x_t + b_f) \quad (2.1)$$

Dimana :

$f_t$ : *forget gate*

$\sigma$ : fungsi aktivasi sigmoid

$W_{hf}$ : *weight* atau bobot *hidden* pada *forget gate*

$h_{t-1}$ : *hidden state* dari *time step* sebelumnya

$W_{xf}$ : *weight* atau bobot *input* pada *forget gate*

$x_t$ : input *time step* saat ini

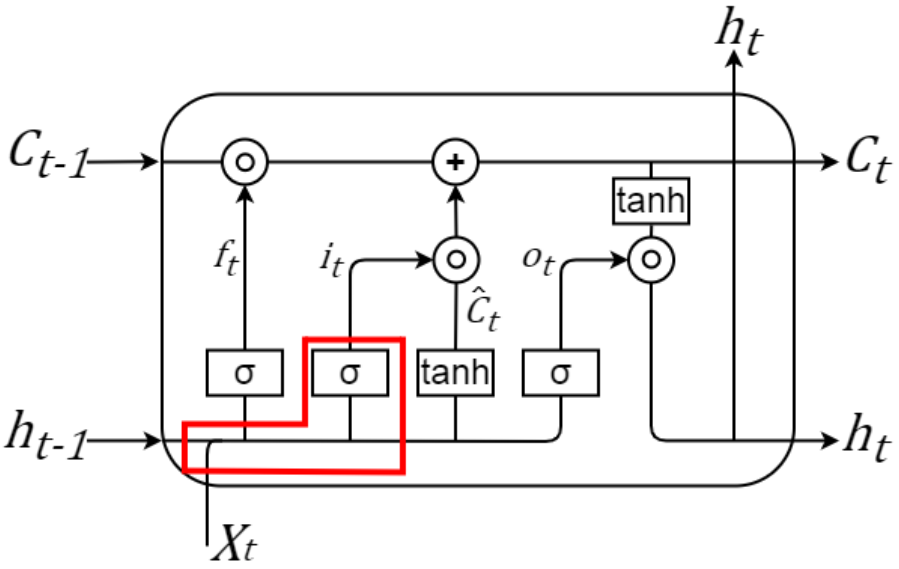
$b_f$ : *bias* pada *forget gate*

*Forget gate* nantinya akan dikalikan dengan *cell state* dari *time step* sebelumnya. Jika *forget gate* bernilai 0 maka *cell state* dari *time step* sebelumnya akan dilupakan atau dibuang. Akan tetapi kika *forget gate* bernilai 1 maka *cell state* dari *time step* sebelumnya akan dilanjutkan ke *time step* saat ini.

*Gate* berikutnya yang terdapat pada LSTM yaitu *Input gate*. *Input gate* memiliki peran untuk mengatur informasi baru yang berasal dari *input* pada *time step* saat ini yang akan dimasukkan dan diproses pada memori jangka panjang sel LSTM. Diagram LSTM untuk *input gate* dapat diperhatikan pada Gambar 2. 5.

*Input gate* nantinya akan proses lagi dan akan ditambahkan ke *cell state baru.* Sehingga *input gate* merepresentasikan informasi yang akan diproses

menjadi *cell state* baru. Untuk menghitung *Input gate* dapat dilakukan dengan menggunakan Persamaan (2.2).




**Gambar 2. 5. Input Gate**

$$i_t = \sigma(W_{hi} \bullet h_{t-1} + W_{xi} \bullet x_t + b_i) \quad (2.2)$$

Dimana :

$i_t$ : *input gate*

$\sigma$ : fungsi aktivasi sigmoid

$W_{hi}$ : *weight* atau bobot *hidden* pada input gate

$h_{t-1}$ : hidden state *time step* sebelumnya

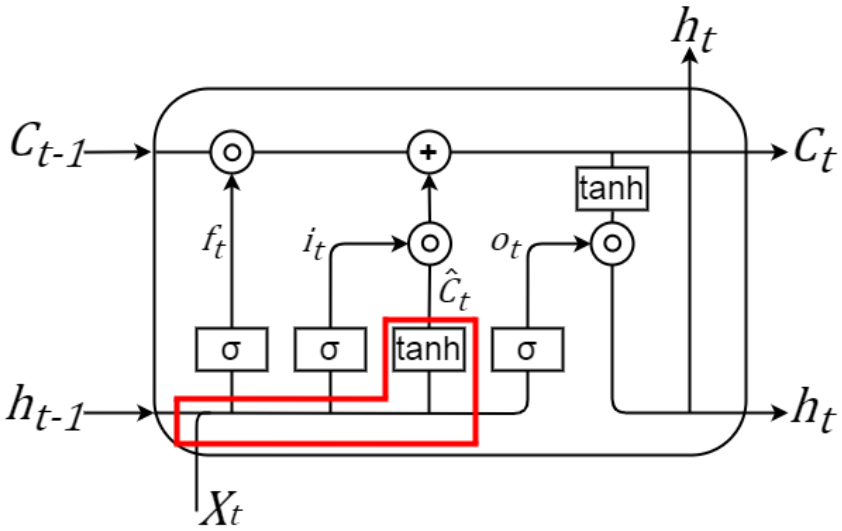
$W_{xi}$ : *weight* atau bobot *input* pada *input gate*

$x_t$ : input *time step* saat ini

$b_i$ : *bias* pada *input gate*

Pada LSTM selain terdapat 3 *gate* utama, terdapat juga *cell state candidate* atau calon *cell state* baru. *Cell state candidate* ini nantinya akan dikalikan per elemen (*hadamard product*) dengan nilai hasil dari *input gate* dan akan diproses untuk mendapatkan cell state baru. Diagram LSTM untuk *cell state candidate* dapat diperhatikan pada Gambar 2. 6.

Untuk menghitung *Cell state candidate* dapat dilakukan dengan menggunakan Persamaan (2.3).




**Gambar 2. 6. Cell State Candidate**

$$\widehat{C}_t = tanh(W_{hc} \bullet h_{t-1} + W_{xc} \bullet x_t + b_c) \quad (2.3)$$

Dimana :

$\widehat{C}_t$ : *cell state candidate*

$tanh$ : fungsi aktivasi tangen hiperbolik

$W_{hc}$ : *weight* atau bobot *hidden* pada *cell state*

$h_{t-1}$ : *hidden state time step* sebelumnya

$W_{xc}$ : *weight* atau bobot *input* pada *cell state*

$x_t$ : input *time step* saat ini

$b_c$ : *bias* pada *cell state*

Hasil dari *cell state candidate* akan dikalikan per elemen dengan *input gate* pada perhitungan *Cell state* baru. C*ell state* baru dapat dihitung dengan menggunakan Persamaan (2.4).

$$C_t = f_t \circ C_{t-1} + i_t \circ \widehat{C}_t \quad (2.4)$$

Dimana :

$C_t$ : *Cell state* baru

$f_t$ : *forget gate*

$C_{t-1}$ : *cell state* time step sebelumnya

$i_t$ : *input gate*

$\tilde{C}_t$ : *cell state candidate*

Pada perhitungan untuk menghitung *cell state* baru, hasil dari *forget gate* akan dikalikan per elemen dengan *cell state* dari *time step* sebelumnya. Dengan

demikian jika *forget gate* memiliki nilai 0, maka *cell state* dari *time step* sebelumnya akan dibuang atau dilupakan, sedangkan jika *forget gate* memiliki nilai 1 maka *cell state* dari *time step* sebelumnya akan diteruskan ke *cell state* pada *time step* saat ini. *Cell state* tersebut nantinya akan diproses untuk menjadi *hidden state* pada *time step* saat ini.

Nilai o*utput* yang nantinya akan menjadi *hidden state* diatur pada *output gate*. *Output gate* dapat dihitung dengan menggunakan Persamaan (2.5).

$$o_t = \sigma(W_{ho} \bullet h_{t-1} + W_{xo} \bullet x_t + b_o) \quad (2.5)$$

Dimana :

$o_t$ : *output gate*

$\sigma$ : fungsi aktivasi sigmoid

$W_{ho}$ : *weight* atau bobot hidden pada output gate

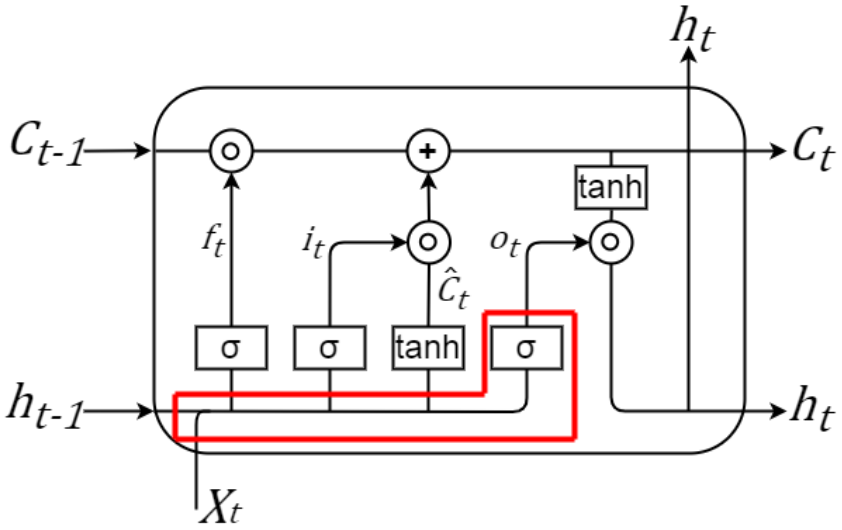
$h_{t-1}$ : *hidden state time step* sebelumnya

$W_{xo}$ : *weight* atau bobot *input* pada *output gate*

$x_t$ : *input time step* saat ini

$b_o$ : *bias* pada *input gate*

Hasil dari *output gate* nantinya akan digunakan untuk menghitung *output*. *Output* ini juga digunakan sebagai *hidden state* untuk *time step* saat ini. Diagram LSTM untuk *output gate* dapat dilihat pada Gambar 2. 7.




**Gambar 2. 7. Output Gate.**

*Output* dihitung dengan mengalikan *output gate* dengan hasil *tangen hiperbolik* dari *cell state* baru dengan menggunakan perkalian antar elemen. *Output* dapat dihitung dengan menggunakan Persamaan (2.6).

$$h_t = o_t \circ \tanh(C_t) \quad (2.6)$$

Dimana :

$h_t$ : *output* atau *hidden state*

$o_t$ : *output gate*

$tanh$ : fungsi tangen hiperbolik

$C_t$ : *cell state time step* saat ini

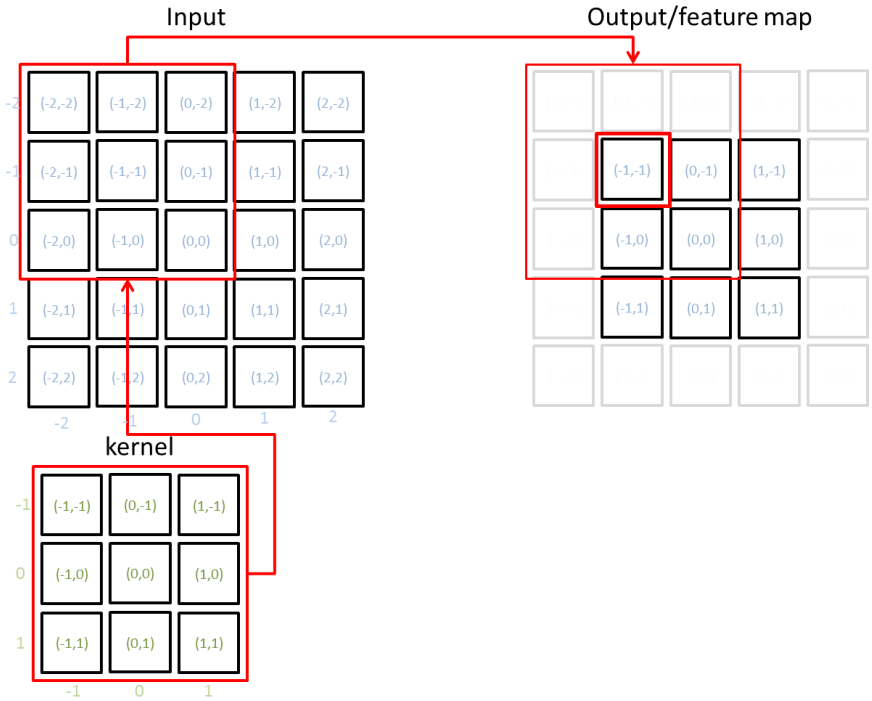
*Gates* yang terdapat pada LSTM memiliki kontrol terhadap informasi yang dapat disimpan atau diabaikan, sehingga menjadikannya metode yang lebih mumpuni untuk mempelajari hubungan kompleks dalam data [24]. Jaringan LSTM dapat digunakan untuk mempelajari karakteristik, mengidentifikasi, dan mengenali peristiwa secara otomatis. LSTM terbukti efektif untuk mempelajari ketergantungan jarak jauh dalam berbagai penelitian sebelumnya [9]. Selain itu LSTM memberikan hasil yang lebih tepat dan pelatihan yang lebih cepat jika dibandingkan dengan *recurrent learning* dan *retro-propagation* dari waktu ke waktu [23].

## 2.5. Konvolusi

Operasi konvolusi pada citra merupakan sebuah operasi yang berguna untuk mendapatkan dan mengolah fitur spatial pada citra. Konvolusi bekerja dengan mengalikan kernel atau filter konvolusi dengan citra *input* dan digeser secara berurutan hingga melintasi seluruh citra *input* tersebut. Hasil dari perkalian kernel dengan citra *input* tersebut dijumlahkan sehingga menghasilkan nilai tunggal yang nantinya menghasilkan *feature map*. Representasi dari proses konvolusi dapat diperhatikan pada Gambar 2. 8.

Pada citra dengan 3 *channel* seperti RGB, proses konvolusi memerlukan 3 kernel juga. Dalam *Keras* jika kita mengatur jumlah kernel sama dengan 1, maka di dalam arsitekturnya secara tidak terlihat *Keras* secara otomatis membuatnya menjadi 3 channel dan begitu juga kelipatannya. Hasil perkalian titik dari seluruh *channel* dengan kernel akan menghasilkan 1 nilai. Sehingga meskipun gambar memiliki 3 *channel*, jumlah *features map* yang dihasilkan tetap 1 (sesuai jumlah kernel). Pada proses konvolusi nilai yang diperbarui selama pelatihan adalah nilai dari kernelnya. Nilai-nilai yang diperbarui selama pelatihan tersebut disebut juga dengan istilah parameter. Dikarenakan parameter tersebut merupakan nilai dari

kernelnya maka jumlah parameter dipengaruhi oleh ukuran kernel, jumlah kernel, dan jumlah channel dari citra input. Untuk menghitung jumlah parameter dan jumlah komputasi dapat dilakukan menggunakan Persamaan (2.8) dan Persamaan (2.9).




**Gambar 2. 8. Konvolusi**

Output atau feature map dari operasi konvolusi dapat diperhatikan pada Persamaan (2.7).

$$f(x,y) = k * i(x,y) = \sum_{dx=-m}^{m} \sum_{dy=-m}^{m} k(dx,dy) \times i(x+dx, y+dy) \qquad (2.7)$$

Dimana:

$f$ : output atau feature map

$k$ : kernel

$i$ : citra input

$x, y$ : koordinat untuk citra input dan output

$dx, dy$ : koordinat untuk kernel

$m$ : batas koordinat kernel

$$Parameter = K_h \times K_w \times C_{in} \times C_{out} \qquad (2.8)$$

$$Jumlah\ komputasi = K_h \times K_w \times C_{in} \times C_{out} \times H \times W \qquad (2.9)$$

Dimana:

$K_h$ : tinggi kernel

$K_w$ : lebar kernel

$C_{in}$ : *channel* citra *input*

$C_{out}$ : *channel output*/jumlah kernel

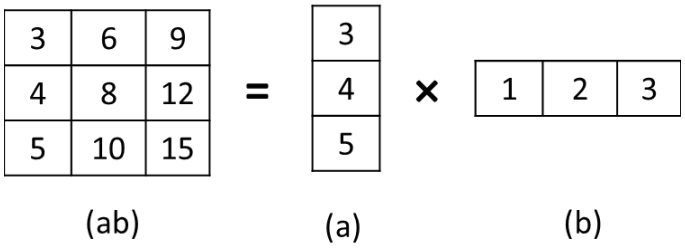
$H$ : tinggi citra *input*

$W$ : lebar kernel

CNN merupakan salah satu contoh implementasi operasi konvolusi pada *neural network*. CNN atau *Convolutional Neural Network* adalah arsitektur jaringan syaraf tiruan yang menggunakan lapisan konvolusi. CNN banyak dimanfaatkan sebagai arsitektur yang digunakan untuk mendapatkan atau mengekstrak fitur spatial dari data citra. *Layer* konvolusi yang terdapat pada CNN dinilai dapat mengurangi beban komputasi dari arsitektur *neural network* [25]. Meskipun tidak memerlukan tahap ekstraksi fitur, CNN juga memiliki performa yang dikatakan bagus [26]. Hal tersebut dikarenakan pada CNN sudah terdapat ekstraksi fitur tersendiri [26]. Lapisan konvolusi, fungsi aktivasi, dan lapisan *pooling* yang terdapat pada CNN berperan besar untuk melakukan ekstraksi fitur tersebut [26].

## 2.6. *Separable Convolution*

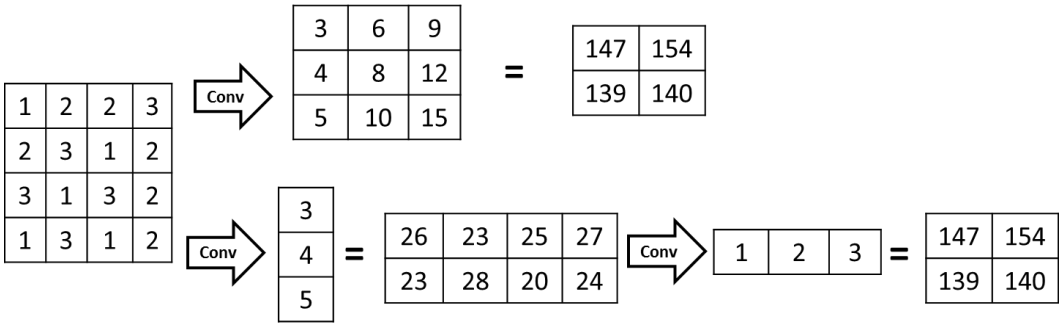
Pada dasarnya, konsep konvolusi dapat dibagi menjadi 2, hal ini disebut *separable convolution*. *Separable convolusion* dilakukan dengan membagi kernel menjadi dua bagian. Contoh matriks kernel yang dibagi dapat diperhatikan pada Gambar 2. 9.




**Gambar 2. 9. Separable Matriks**

Pada Gambar 2. 9 matriks (ab) merupakan matriks 3×3 yang dapat dipisahkan menjadi matriks (a) dan matriks (b). Jika dibalik, matriks (ab) merupakan hasil kali dari matriks (a) dengan matriks (b). Hasil konvolusi suatu

citra *input* dengan matriks (ab) akan sama dengan hasil konvolusi dengan matriks (a) lalu dikonvolusi lagi dengan matriks (b). Contoh perbandingan konvolusi dapat diperhatikan pada Gambar 2.10.




**Gambar 2. 10. Convolution vs Separable Convolution**

Pada Gambar 2. 10 konvolusi dengan *separable convolution* menghasilkan hasil yang sama. Selain itu pada *separable convolution* memiliki lebih sedikit perhitungan dan lebih sedikit parameter. Meskipun demikian, tidak semua matriks kernel dapat dibagi menjadi 2 bagian dengan konsep yang sama.

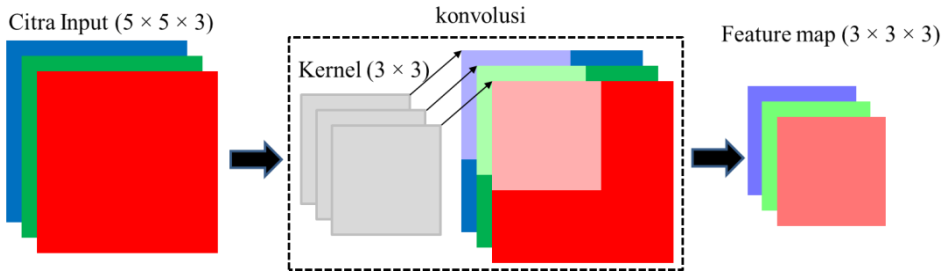
## 2.7. Depthwise Separable Convolution

Terdapat banyak teknik yang dapat digunakan untuk mereduksi parameter pada teknik konvolusi. Salah satunya adalah menggunakan teknik *Depthwise separable convolution*. *Depthwise separable convolution* merupakan jenis konvolusi yang secara efektif mengurangi jumlah penghitungan parameter dari proses konvolusi pada umumnya [17]. Dengan jumlah parameter dan komputasi yang lebih sedikit, arsitektur model dapat melakukan proses pelatihan yang lebih cepat tanpa mengurangi kemampuannya secara signifikan. Pada implementasinya dengan menggunakan data video, Depthwise Separable Convolution digunakan untuk mendapatkan dan meneruskan fitur spatial dari data inputan [27].

*Depthwise separable convolution* bekerja dengan melakukan 2 proses konvolusi, yaitu *depthwise convolution*, dan *pointwise convolution*.

*1. Depthwise convolution*

Pada *depthwise convolution* citra *input* akan dilakukan proses konvolusi tanpa merubah kedalaman. Ilustrasi *depthwise convolution* dengan menggunakan citra *input* 5 × 5 dengan kernel 3 × 3 dapat diperhatikan pada Gambar 2. 11.




**Gambar 2. 11. Depthwise Convolution**

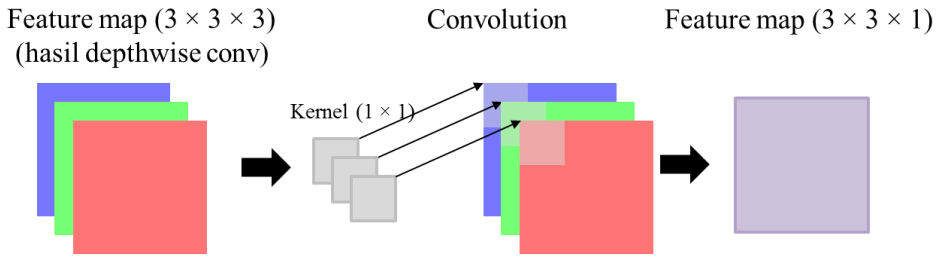
Pada *depthwise convolution* setiap kernel 3 × 3 mengelilingi 1 channel citra untuk mendapatkan scalar product 9 piksel pada setiap grup, sehingga menghasilkan *feature map* 3 × 3 × 1. Features map yang dihasilkan ditumpuk sehingga menghasilkan 3 × 3 × 3 *features map*. Pada *depthwise convolution* jumlah parameter dan jumlah komputasi dapat dihitung dengan menggunakan Persamaan (2.10) dan Persamaan (2.11).

$$Parameter_{DC} = K_h \times K_w \times C_{in} \quad (2.10)$$

$$n\ komputasi_{DC} = K_h \times K_w \times C_{in} \times H \times W \quad (2.11)$$

2. Pointwise convolution

*Pointwise convolution* merupakan proses konvolusi dengan menggunakan kernel 1 × 1. Konvolusi dengan kernel 1 × 1 digunakan untuk merubah kedalaman *Feature Map*. Jumlah kernel dapat disesuaikan dengan kebutuhan. Jumlah kernel ini akan sama dengan jumlah output yang dihasilkan. Ilustrasi *pointwise convolution* dengan menggunakan citra *input* hasil dari *depthwise convolution* dengan 1 kernel 1 × 1 dapat diperhatikan pada Gambar 2. 12.



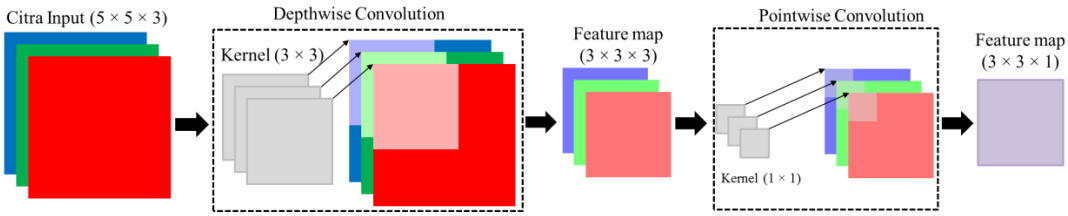

**Gambar 2. 12. Pointwise Convolution**

Untuk menghitung jumlah parameter dan jumlah komputasi pada *depthwise convolution* dapat dengan menggunakan Persamaan (2.12) dan Persamaan (2.13).

$$Parameter_{PC} = C_{in} \times C_{out} \times K_h \times K_w \quad (2.12)$$

$$n\ komputasi_{PC} = C_{in} \times C_{out} \times H \times W \quad (2.13)$$

Dengan demikian *depthwise separable convolution* menggunakan citra input 5 × 5 dengan kernel 3 × 3 dan jumlah kernel yaitu 1 dapat diilustrasikan pada diagram Gambar 2. 13.




**Gambar 2. 13. Depthwise Separable Convolution**

Dari proses yang dilakukan di atas dapat disimpulkan untuk menghitung jumlah parameter dan jumlah komputasi pada *depthwise separable convolution* dapat dihitung dengan menggunakan Persamaan (2.14) dan Persamaan (2.15).

$$Parameter_{DSC} = Parameter_{DC} + Parameter_{PC} \quad (2.14)$$

$$= (K_h \times K_w \times C_{in}) + (C_{in} \times C_{out} \times K_h \times K_w)$$

$$n\ komputasi_{DSC} = n\ komputasi_{DC} + n\ komputasi_{PC} \quad (2.15)$$

$$= (K_h \times K_w \times C_{in} \times H \times W) + (C_{in} \times C_{out} \times H \times W)$$

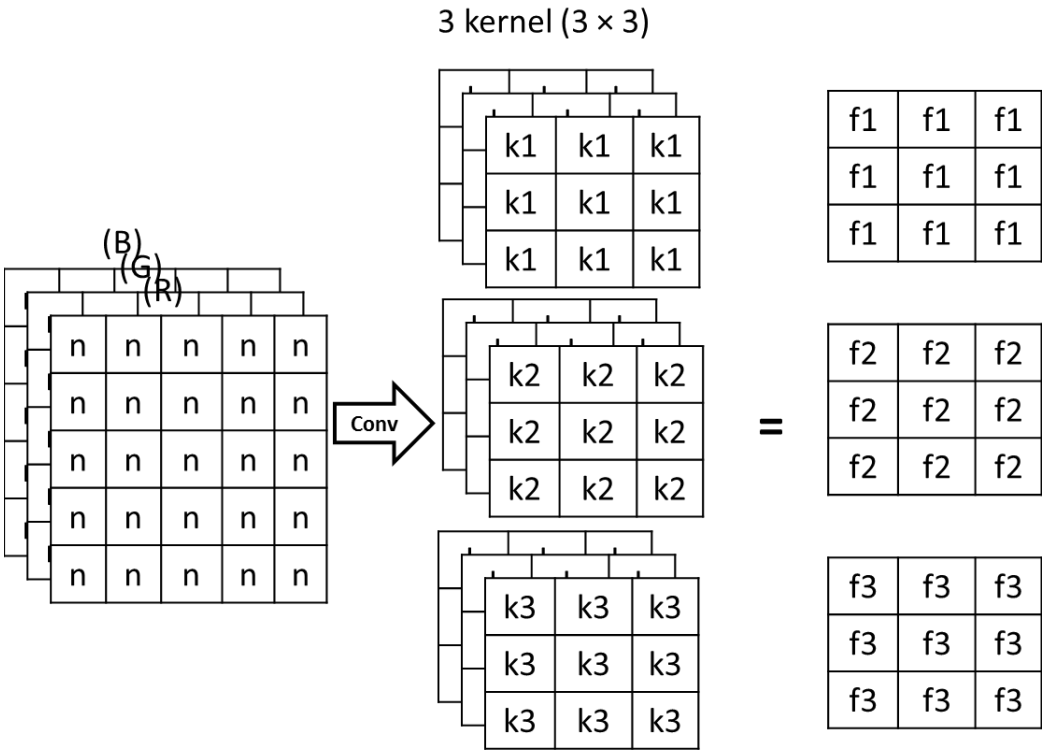
Dari Persamaan (2.14) dan Persamaan (2.15) dapat disimpulkan bahwa *depthwise separable convolution* dapat mereduksi atau mengurasi jumlah parameter dan jumlah komputasi yang dilakukan [16], sehingga meringankan proses pelatihan model.

Sebagai contoh untuk perbedaan konvolusi biasa dengan *depthwise separable convolution* dapat dilihat pada contoh berikut. Misalkan terdapat sebuah citra dengan ukuran 5×5 dengan 3 *channel* yaitu RGB. Citra tersebut akan dilakukan konvolusi dengan 3 kernel 3×3 dengan *stride* 1 dan tanpa *padding*. Ilustrasi proses konvolusi dengan konvolusi biasa dapat diperhatikan pada Gambar 2. 14.

Pada Gambar 2. 14 setiap kernel akan menghasilkan 1 *feature map*, sehingga jika terdapat 3 kernel maka akan menghasilkan 3 *feature map*. Dengan menggunakan Persamaan (2.8) jumlah parameter dari contoh tersebut yaitu :

$$Parameter = 3 \times 3 \times 3 \times 3$$

$$= 81$$




**Gambar 2. 14. Contoh Konvolusi Biasa**

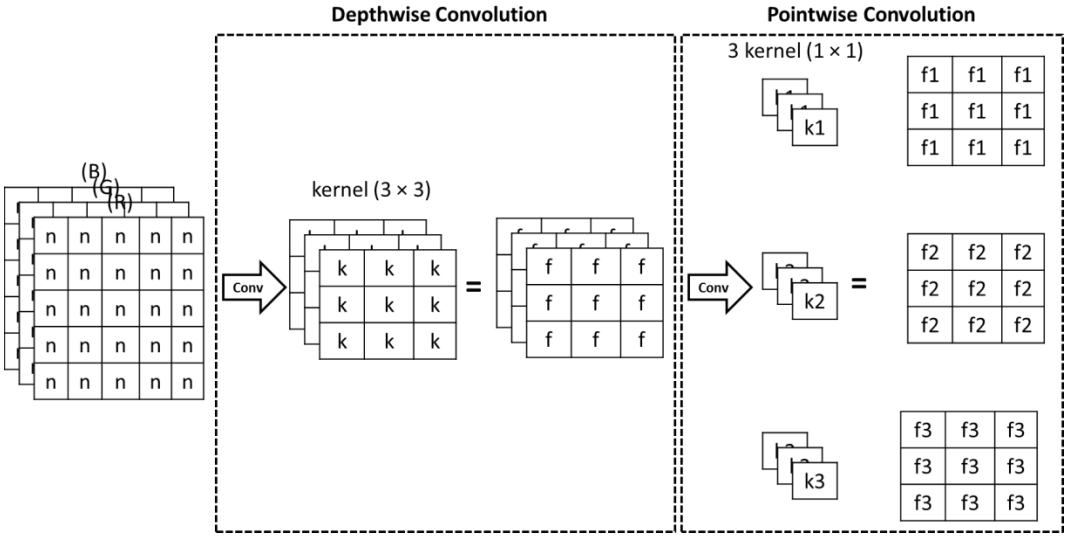
Untuk melihat apakah perhitungan yang dilakukan sudah benar atau tidak, dilakukan pembuktian menggunakan *library keras* dengan membuat *summary model* 1 layer Conv2D dengan menggunakan jumlah dan ukuran kernel yang sama dan tanpa bias. Hasil dari summary model dapat diperhatikan pada Gambar 2. 15.

```
Model: "sequential_5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_2 (Conv2D)           (None, 98, 98, 3)         81

=================================================================
Total params: 81 (324.00 Byte)
Trainable params: 81 (324.00 Byte)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

**Gambar 2. 15. Summary model ConvD 3 kernel 3*3**

Dengan contoh yang sama tetapi proses konvolusi menggunakan *Depthwise Separable Convolution* dapat diperhatikan pada Gambar 2. 16.




**Gambar 2. 16. Contoh Depthwise Separable Convolution**

Pada *depthwise separable convolution* jika kita mengatur berapapun jumlah kernelnya, yang di-*generate* pada *depthwise convolution* hanya 1 kernel. Seperti penjelasan sebelumya, library akan men-*generate channel* dari kernel tersebut sejumlah *channel* dari *citra input.* Sehingga 1 kernel tersebut akan memiliki 3 channel juga. Dengan menggunakan Persamaan (2.10) jumlah parameter pada *depthwise convolution* dari contoh tersebut yaitu :

$$Parameter_{DC} = 3 \times 3 \times 3$$
$$= 27$$

Jumlah kernel yang diatur akan berpengaruh pada *pointwise convolution.* Seperti penjelasan sebelumya, library akan men-*generate channel dari* kernel tersebut sejumlah *channel* dari *citra input*. Dari kernel tersebut nantinya akan menghasilkan 1 *feature map*. Pada contoh tersebut, jumlah kernel yang diatur adalah 3, sehingga nantinya setiap kernel akan memiliki 3 *channel*, dan *feature map* yang dihasilkan yaitu 3. Dengan menggunakan Persamaan (2.12) jumlah parameter pada *pointwise convolution* dari contoh tersebut yaitu :

$$Parameter_{PC} = 3 \times 3 \times 1 \times 1$$
$$= 9$$

Sehingga dengan menggunakan Persamaan (2.14) jumlah parameter pada proses keseluruhan *depthwise separable convolution* yaitu :

$$Parameter_{DSC} = 27 + 9$$
$$= 36$$

Untuk melihat apakah perhitungan yang dilakukan sudah benar atau tidak, dilakukan juga pembuktian menggunakan *library keras* dengan membuat *summary model* 1 layer SeparableConv2D dengan menggunakan jumlah dan ukuran kernel yang sama dan tanpa bias. Hasil dari summary model dapat diperhatikan pada Gambar 2. 17.

```
Model: "sequential_6"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 separable_conv2d_2 (Separa  (None, 98, 98, 3)         36
 bleConv2D)

=================================================================
Total params: 36 (144.00 Byte)
Trainable params: 36 (144.00 Byte)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

**Gambar 2. 17. Summary model SeparableConv2D 3 kernel 3*3**

Dari contoh tersebut, dengan menggunakan kasus yang sama, kedua proses konvolusi menghasilkan jumlah *feature map* dan ukuran *feature map* yang sama, akan tetapi jumlah parameter yang dihasilkan berbeda. Jumlah parameter pada *depthwise separable convolution* jauh lebih sedikit daripada jumlah parameter pada konvolusi biasa.

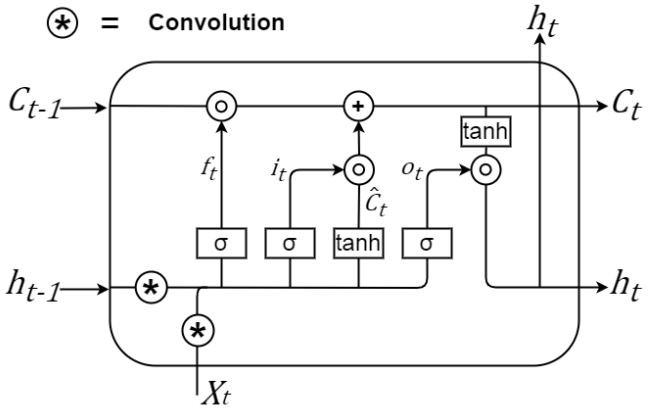
## 2.8. *Convolutional* LSTM

Untuk menangani dan memproses data dengan urutan waktu seperti pada data berurutan atau *time series*, arsitektur yang cocok digunakan yaitu LSTM [17], dikarenakan LSTM merupakan arsitektur RNN yang dinilai dapat mengatasi permasalahan korelasi waktu. Akan tetapi pada implementasinya, LSTM tidak memungkinkan untuk menerima *input* data spatial seperti citra secara langsung. Citra *input* sebelumnya harus dilakukan ekstraksi fitur dan diubah dimensinya sedemikian rupa agar dapat diproses atau diolah oleh LSTM. Hal tersebut mengakibatkan data yang diproses pada LSTM sudah tidak mempertahankan fitur spatialnya.

Video merupakan gabungan dari beberapa *frame* yang berupa citra. Untuk mengolah citra, arsitektur yang umum digunakan adalah *Convolutional Neural Network* atau CNN [26]. CNN merupakan arsitektur jaringan syaraf tiruan yang menggunakan lapisan konvolusi. Lapisan konvolusi pada CNN sangat berperan penting dalam memproses citra.

Setiap *frame* pada video memiliki keterkaitan antara *frame-frame* sebelumnya. Sehingga dapat dikatakan bahwa data video merupakan data dengan fitur *spatio-temporal*. Untuk memproses data dengan fitur *spatio temporal* seperti pada data video, arsitektur LSTM dapat memanfaatkan operasi konvolusi yang diterapkan pada arsitekturnya. Arsitektur ini disebut *Convolutional* LSTM. *Convolutional* LSTM merupakan kombinasi antara LSTM dengan operasi konvolusi, sehingga dapat menerima dan memproses data dengan fitur *spatio-temporal*. *Convolutional* LSTM terkenal untuk mengekstraksi fitur *spatio-temporal* yang melekat pada input dengan jangkauan luas [28]. *Convolutional* LSTM dinilai dapat menangkap informasi *temporal* dan *spatial* dan memberikan kinerja superior pada *frame* video dibandingkan dengan *fully connected* LSTM [11][12].

*Convolutional* LSTM hampir sama dengan LSTM biasa, yang membedakan adalah perkalian matriks pada LSTM diganti dengan operasi konvolusi pada setiap *gates* [14][17]. Diagram *Convolutional* LSTM dapat dilihat pada Gambar 2. 18 dimana "*****" merupakan operasi konvolusi




**Gambar 2. 18. Convolutional LSTM**

Dari Gambar 2. 18 dapat disimpulkan bahwa persamaan untuk untuk *Convolutional* LSTM menjadi seperti pada Persamaan (2.16) sampai Persamaan (2.21).

$$i_t = \sigma(W_{hi} * h_{t-1} + W_{xi} * x_t + b_i) \quad (2.16)$$

$$f_t = \sigma(W_{hf} * h_{t-1} + W_{xf} * x_t + b_f) \quad (2.17)$$

$$\widehat{C_t} = tanh(W_{hc} * h_{t-1} + W_{xc} * x_t + b_c) \quad (2.18)$$

$$C_t = f_t \circ C_{t-1} + i_t \circ \widehat{C}_t \quad (2.19)$$

$$o_t = \sigma(W_{ho} * h_{t-1} + W_{xo} * x_t + b_o) \quad (2.20)$$

$$h_t = o_t \circ \tanh(C_t) \quad (2.21)$$

Dimana :

* = operasi konvolusi

Jika dibandingkan dengan LSTM biasa, perbedaanya terletak pada perkalian matriks pada setiap gates digantikan dengan operasi konvolusi.

Jumlah parameter pada *Convolutional LSTM* didapatkan dari total nilai pada kernel dari proses konvolusi pada setiap *gate*. Pada setiap *gate* terdapat kernel untuk input *convolution* yang disimbolkan sebagai $W_x$ dan kernel untuk *recurrent convolution* yang disimbolkan sebagai $W_h$. Jumlah parameter pada $W_x$ didapatkan dari ukuran kernel dikali jumlah *channel* dari citra *input* dikali jumlah kernel yang digunakan. Sedangkan jumlah parameter pada $W_h$ didapatkan dari ukuran kernel dikali jumlah kernel dikali ukuran *channel* dari citra output. Pada *Convolutional* LSTM terdapat 4 gate sehingga jumlah total parameter didapatkan dari hasil jumlah parameter pada $W_x$ dan $W_h$ dikali 4. Singkatnya untuk menghitung jumlah parameter pada *Convolutional* LSTM dapat menggunakan Persamaan (2.22) atau Persamaan (2.23).

$$Parameter\ \text{ConvLSTM} = [(K_h \times K_w \times C_{in} \times C_{out}) + (K_h \times K_w \times C_{out} \times C_{out})] \times 4 \quad (2.22)$$

$$= [K_h \times K_w \times C_{out} \times (C_{in} + C_{out})] \times 4 \quad (2.23)$$

Jumlah parameter *Convolutional* LSTM juga dapat dihitung dengan cara menghitung jumlah parameter pada $W_x$ dan $W_h$ dikali 4 seperti pada Persamaan (2.24) sampai dengan Persamaan (2.26):

$$W_x = K_h \times K_w \times C_{in} \times C_{out} \quad (2.24)$$

$$W_h = K_h \times K_w \times C_{out} \times C_{out} \quad (2.25)$$

$$\text{Parameter ConvLSTM} = (W_x + W_h) \times 4 \quad (2.26)$$

Sebagai contoh misalkan terdapat layer Convolutional LSTM dengan jumlah kernel 5, ukuran kernel (3 × 3), padding same, tanpa bias. Sedangkan untuk inputnya merupakan citra RGB. Total parameter dapat dihitung dengan Persamaan (2.23):

$$\text{Parameter ConvLSTM} = [K_h \times K_w \times C_{out} \times (C_{in} + C_{out})] \times 4$$

$$= [3 \times 3 \times 5 \times (3 + 5)] \times 4 = 1440$$

Selain itu untuk menghitung parameter Convolutional LSTM dapat juga menggunakan Persamaan (2.24) sampai dengan Persamaan (2.26).

$$W_x = 3 \times 3 \times 3 \times 5 = 135$$

$$W_h = 3 \times 3 \times 5 \times 5 = 225$$

$$\text{Parameter ConvLSTM} = (135 + 225) \times 4 = 1440$$

Pada perhitungan parameter pada $W_h$ terdapat perkalian dengan $C_{out}$ sebanyak dua kali yaitu 5 × 5. Nilai ini dikarenakan $h_{t-1}$ memiliki ukuran (width, height, 5) sesuai dengan ukuran output, sedangkan ukuran output akan sama dengan jumlah kernel yang digunakan, pada contoh ini yaitu 5. Jika 5 kernel dikonvolusikan dengan $h_{t-1}$ yang memiliki shape (widht, height, 5) maka setiap kernel akan men-generate menjadi 5 kernel (sama seperti konsep konvolusi untuk input beberapa channel) sehingga total kernel yang digenerate yaitu 5 × 5 = 25.

## 2.9. Persentase Penurunan

Persentase penurunan digunakan untuk melihat atau mengetahui seberapa besar perbandingan dari suatu nilai ke nilai yang lebih rendah. Persentase penurunan dapat dihitung dengan menggunakan Persamaan (2.27).

$$Persentase\ Penurunan = \left(\frac{Nilai\ tinggi - Nilai\ rendah}{Nilai\ tinggi}\right) \times 100\% \qquad (2.27)$$

## 2.10. Evaluasi Kinerja Sistem

Evaluasi sistem merupakan tahapan penting untuk mengidentifikasi kinerja dari model yang telah latih menggunakan algoritma tertentu. Dalam mendapatkan model yang terbaik, dibutuhkan parameter uji yang digunakan sebagai pembanding dalam suatu model. Salah satu metode untuk evaluasi model adalah dengan menggunakan *Confusion Matix* Penelitian ini menggunakan dataset dengan 2 kelas sehingga proses evaluasi yang digunakan adalah *binary classification.* Ada beberapa karakteristik dalam penentuan klasifikasi.

1. *Confusion matrix*

*Confusion matrix* adalah salah satu metode yang dapat digunakan untuk merepresentasikan hasil prediksi dari suatu dataset yang telah di uji. *Confusion Matrix* terdiri dari empat komponen. Pertama, *True Positive* (TP) yaitu ketika hasil prediksi positif dan kelas sebenarnya adalah positif. Kedua, *False Positive*

(FP) yaitu ketika hasil prediksi positif sedangkan kelas sebenarnya adalah negatif. Ketiga, *True Negative* (TN) yaitu ketika hasil prediksi negatif dan kelas sebenarnya adalah negatif. Terakhir, *False Negative* (FN) yaitu ketika model memprediksi negative sedangkan kelas sebenarnya adalah positif [29]. Gambaran *Confusion Matrix* yang akan digunakan pada penelitian ini dapat diperhatikan pada Gambar 2. 19.

| | | Hasil Prediksi | |
|---|---|---|---|
| | | Mengantuk | Tidak Mengantuk |
| Kelas Sebenarnya | Mengantuk | **TP** | **FN** |
| | Tidak Mengantuk | **FP** | **TN** |

**Gambar 2. 19. Confusion matrik**

2. *Accuracy*

*Accuracy* digunakan untuk mengetahui perbandingan prediksi yang benar baik kelas positif maupun kelas negatif dengan seluruh data pelatihan [29]. *Accuracy* dapat dihitung dengan Persamaan (2.24).

$$Accuracy = \frac{Banyak\ prediksi\ benar}{Total\ data\ yang\ diprediksi} \tag{2.28}$$

$$= \frac{TP + TN}{TP + FP + TN + FN}$$

Dimana :

TP : *True Positive*

FP : *False Positive*

TN: *True Negative*

FN: *False Negative*

3. *Precision*

*Precision* atau presisi digunakan untuk mengetahui tingkat ketepatan dari hasil prediksi positif dan kelas sebenarnya positif terhadap seluruh hasil prediksi positif. Untuk mendapatkan nilai *Precision* dapat dihitung dengan Persamaan (2.25).

$$Precision = \frac{TP}{TP + FP} \tag{2.29}$$

4. *Recall*

Digunakan untuk mengetahui sejauh mana model yang telah dibuat dapat mengidentifikasi data pada kelas yang sama. Hal ini dilakukan dengan jumlah data positif yang berhasil diprediksi oleh model dengan total data dengan kelas sebenarnya positif [29]. Untuk mendapatkan nilai *Recall* dapat dihitung dengan Persamaan (2.26).

$$Recall = \frac{TP}{TP + FN} \tag{2.30}$$

5. *F1-Score*

*F1-Score* adalah matriks yang digunakan untuk menghitung *harmonic-mean* dari *precision* dan *recall*. Nilai *F1-Score* memiliki rentang 0 hingga 1, dengan nilai terbaik 1 dan nilai terburuk 0. F1-score digunakan untuk mengidentifikasi model memiliki *precision* dan *recall* yang baik. Persamaan *F1-Score* dihitung dengan penggabungan *precision* dan *recall* dalam satu metrik yang disimbolkan dengan rentan nilai 0 sampai dengan 1 [29]. Untuk menghitung *F1-Score* dapat dihitung dengan Persamaan (2.27).

$$F1\ Score = 2 * (\frac{Recall * Precision}{Recall + Precision}) \tag{2.31}$$

## 2.11. Penelitian Terkait

Pada tahun 2018 terdapat penelitian yang dilakukan oleh Zhang *et. al.* [13] tentang penerapan *attention* pada *Convolutional* LSTM untuk pengenalan gerakan. *Dataset* yang digunakan ada 2, yaitu *jester* yang merupakan Dataset gerakan tangan manusia dan *IsoGD* yang merupakan *Dataset* gerakan manusia. *Dataset* tersebut berupa video. Pada penelitian ini terdapat 4 varian metode yang digunakan yaitu, pertama, dengan menghapus struktur *convolutional* pada *gates*

dan menerapkan *depthwise separable convolution* untuk mereduksi parameter. Kedua, dengan menerapkan mekanisme *attention* pada *input*-nya. Ketiga, dengan merekonstruksi *input gate* menggunakan *channel-wise attention*. Keempat, dengan merekonstruksi output gate dengan *channel-wise attenton.* Dari penelitian yang dilakukan didapatkan beberapa nilai akurasi. Untuk *Dataset jester* menggunakan ConvLSTM biasa mendapatkan akurasi 96,11%, varian pertama 95,12%, varian kedua 94,18%, varian ketiga 95,13%, dan varian keempat 95.10%. Sedangkan untuk *Dataset IsoGD* dengan menggunakan ConvLSTM biasa mendapatkan akurasi 52,01%, varian pertama 55,98%, varian kedua 43,93%, varian ketiga 53,27%, dan varian keempat 54,11%.

Pada penelitian tahun 2019 oleh You, *et. al.* [5]. Penelitian ini membahas tentang deteksi kantuk berkendara secara *real-time* dengan mempertimbangkan perbedaan individu. Penelitian ini dilakukan dengan beberapa tahapan yaitu ekstraksi *frame*, deteksi wajah menggunakan *Deep Cascaded Convolutional Neural Network* atau DCCNN, dan pemberian *landmarks* wajah. Dari *landmarks* wajah didapatkan *Eyes Aspect Ratio* atau EAR. Dari EAR inilah dilakukan proses pelatihan menggunakan *Support Vector Machine* atau SVM. Dari proses pelatihan dan evaluasi yang dilakukan didapatkan akurasi 94,80%.

Pada tahun 2019 terdapat penelitian yang dilakukan oleh Ye *et. al.* [14] tentang pengenalan aktivitas manusia dengan menggunakan *Two-Stream Convolutional Network* dan *Convolutional* LSTM. *Dataset* yang digunakan yaitu UCF101 dan HMDB51. Dataset ini berupa video sehingga dibutuhkan ekstraksi *frame.* Pada penelitian ini fitur spatial dan temporal dilakukan ekstraksi fitur secara terpisah sehingga terdapat 2 aliran. Aliran pertama digunakan untuk mengekstrak fitur spatial. Aliran ini menggunakan input *frame* RGB dan dilakukan ekstraksi fitur dengan arsitektur ResNet-101. Sedangkan aliran kedua digunakan untuk mengekstrak fitur temporal. Aliran ini menggunakan input frame *Stacked Optical Flow* lalu dilakukan ekstraksi fitur dengan ResNet-101. Data hasil ekstraksi fitur pada setiap aliran tersebut akan disatukan dan dilakukan pelatihan dengan menggunakan Convolutional LSTM. Penelitian ini menghasilkan akurasi 69,4% untuk Dataset HMDB51 dan 93,9% untuk Dataset UCF-101.

Pada tahun 2019 juga terdapat penelitian yang dilakukan oleh Duman dan Erdem [11] tentang deteksi anomali pada video. Penelitian ini menggunakan beberapa Dataset yaitu Avenue, UCSD Ped1, and UCSD Ped2. Dataset tersebut berupa video sehingga diperlukan ekstraksi frame terlebih dahulu dan diubah ukurannya. Selanjutnya setiap frame diubah menjadi *Grayscale.* Selanjutnya data tersebut dilakukan ekstraksi fitur dengan *optical flow.* Setelah itu data hasil ekstraksi fitur dilakukan pelatihan dengan menggunakan Convolutional Autoencoder and Convolutional LSTM. Dari penelitian yang dilakukan didapatkan AUC sebesar 89,5% untuk Dataset avenue, 92,4% untuk Dataset Ped1, dan 92,9% untuk Dataset Ped2

Selain itu terdapat juga penelitian pada tahun 2021 oleh Altameem, *et. al.* [6]. Penelitian ini tentang identifikasi dan deteksi dini kantuk pengemudi dengan *hybrid machine learning*. Penelitian ini dilakukan dengan melalui beberapa tahapan yaitu, ekstraksi *frame*, deteksi wajah, lalu dilakukan monitoring mata. Proses pelatihan dilakukan dengan menggunakan SVM. Proses pengujian atau evaluasi dilakukan pada beberapa skenario. Skenario pertama pada saat siang hari dengan kondisi normal memiliki akurasi 91%, skenario kedua pada siang hari dengan pencahayaan redup memiliki akurasi 81%, skenario ketiga pada malam hari dengan pencahayaan terang menghasilkan akurasi 93%, dan skenario 4 pada malam hari dengan pencahayaan redup menghasilkan akurasi 68%.

Pada tahun 2022 Sharma, *et. al.* [30] melakukan penelitian tentang deteksi kelelahan dan sistem kewaspadaan kantuk secara *real time* berbasis jaringan *deep convolutional.* Penelitian ini dilakukan dengan beberapa tahap, yaitu mendapatkan *frame* pengemudi, deteksi wajah, deteksi mata dan deteksi menguap lalu dilakukan klasifikasi dengan menggunakan SVM. Penelitian ini memiliki beberapa persamaan dengan penelitian yang dilakukan  oleh oleh Altameem, *et. al.* [6] pada tahun 2021. Perbedaannya hanya terletak pada implementasi *Deep Convolutional* sebelum dilakukan klasifikasi dengan menggunakan SVM. Penelitian ini menghasilkan rata-rata akurasi 83,25% pada semua skenario.

Pada tahun 2022 Mousavikia *et. al.* [31] melakukan penelitian tentang desain dan implementasi sistem deteksi kantuk pengemudi menggunakan *processor* RiscV yang dimodifikasi *Field-Programmable Gate Array* (FPGA)

(FPGA). Dataset yang digunakan merupakan dataset video sehingga perlu dilakukan ektraksi *frame* sebanyak 1 *frame* per detik. Metode yang digunakan pada penelitian ini yaitu *Convolutional Neural Network* (CNN) yang diimplementasikan pada *processor* RiscV. Model dilatih untuk mengklasifikasikan empat ekspresi utama pengemudi yaitu normal, gangguan, menguap, dan tidur. Akurasi CNN yang dilatih adalah 81,07% pada data validasi.

Pada tahun 2022 Yılmaz dan Akcayol [32] melakukan penelitian tentang deteksi kantuk pengemudi dan memperingatkan pengemudi dengan memeriksa situasi pengemudi atau mengemudi. Penelitian ini menggunakan *Dataset* SUST-DDD atau *Sivas University of Science and Technology Driver Drowsiness Dataset. Dataset* berupa video sehingga perlu dilakukan ekstraksi *frame*. Dari setiap data video dilakukan ekstraksi menjadi 20 frame. Pada penelitian ini pelatihan dilakukan menggunakan model pembelajaran mendalam seperti AlexNet, VGG16, VGG19 dan VGGFaceNet. *Outputs* yang diperoleh dari pelatihan tersebut dimasukkan ke dalam arsitektur LSTM untuk proses klasifikasi. Penelitian ini menghasilkan beberapa hasil yaitu arsitektur VGG19+LSTM menghasilkan akurasi 90,53%, VGG16+LSTM 89,39%, AlexNet+LSTM 63,91%, dan VGGFaceNet+LSTM 84,94%.

Pada tahun 2023 Vijaypriya dan Uma [33] melakukan penelitian tentang deteksi kantuk berbasis fitur wajah dengan *Multi-Scale Convolutional Neural Network* (MCNN). Pada penelitian ini untuk melakukan pelatihan model menggunakan *Dataset* dari *Yawning Detection Dataset* (YAWDD) dan NTHU-DDD. Penelitian ini dilakukan dengan beberapa tahapan yaitu merubah video menjadi *frame*, setelah itu dilakukan ekstraksi *key frame* dan seleksi *frame*. Selanjutnya dilakukan deteksi wajah dan pemberian *landmarks*. Sebelum dilakukan pelatihan, dilakukan *preprocessing* data menggunakan *Cross-Guided Bilateral Filtering.* Data hasil *preprocessing* dilakukan ekstraksi fitur. Setelah itu dilakukan pelatihan dengan menggunakan MCNN dengan *Flamingo Search Algorithm* (FSA). Jika pengemudi menguap atau mata tertutup maka diklasifikasikan sebagai mengantuk, sedangkan jika Mata terbuka, mulut tertutup, atau berbicara diklasifikasikan sebagai tidak mengantuk. Dari hasil evaluasi

didapatkan akurasi 98,38% untuk *Dataset* YAWDD dan 98,26% untuk NTHU-DDD.

Pada penelitian yang dilakukan oleh Abdullah *et. al.* [19] yang dilakukan pada tahun 2023, penelitian tentang deteksi kelelahan pengemudi dilakukan dengan menggunakan *Raspberry-Pi*. Dari dataset yang ada dilakukan identifikasi 68 *facial landmarks*. Dari hasil identifikasi *face landmark* tersebut akan didapatkan fitur mata dan mulut. Metode yang digunakan yaitu *Deep Learning*, ANN, dan EAR (*Eye Aspect Ratio*). Metode *deep learning* menghasilkan rata-rata akurasi 73,67%, metode ANN 88%, dan EAR menghasilkan akurasi 90 dan 100%.

Pada tahun 2023, Kim *et.al.* [27] melakukan penelitian tentang antisipasi action dalam game. Untuk mengantisipasi suatu aksi dalam game diperlukan pengenalan aksi dalam game tersebut sehingga dapat dilakukan antisipasi. Penelitian ini menggunakan Dataset tayangan ulang StarCraft II saat Terrans menang pada pertandingan Terrans vs Protoss. Metode yang digunakan yaitu Depthwise Separable Convolution dan LSTM. Penelitian ini berfokus pada penggunaan 2 modal dalam pelatihan model, yaitu label dan frame video. Frame video yang telah dilakukan preprocessing masuk ke Depthwise Separable Convolution untuk ekstraksi fitur, sedangkan label akan dilakukan label embedding. Selanjutnya kedua modal tersebut dilakukan concat dan dimasukkan ke dalam LSTM. Penelitian ini menghasilkan macro F1-Score 0,4475.

Pada tahun 2023, Mobsite *et.al.* [15] melakukan penelitian tentang klasifikasi aktivitas manusia dan deteksi jatuh. Penelitian ini menggunakan beberapa dataset, yaitu COCO 2017, dan UP-FALL. Dari data video dilakukan deteksi aktivitas, sehingga klasifikasi terjadi hanya ketika seseorang terdeteksi dalam lingkungan. Selanjutnya data hasil deteksi aktivitas tersebut dilakukan ekstraksi fitur dengan menggunakan Global History of Motion (GHM). Setelah itu gambar hasil GHM akan diproses dengan Dilated Convolutional Long Short Term Memory yang dikombinasikan dengan Lightweight Deep Neural Network (LDNN). Selain itu Depthwise Separable Convolution Juga digunakan pada LDNN untuk mereduksi parameter. Penelitian ini menghasilkan F1-Score sebesar

98,46% untuk deteksi aktivitas. Sedangkan untuk deteksi jatuh menghasilkan F1-Score sebesar 98,87%.

Tabel 2.1 Penelitian terkait

| No | Peneliti, Tahun | Permasalahan | Model/Solusi | Hasil |
|---|---|---|---|---|
| 1 | Zhang *et. al.* [13], 2018 | Pengenalan gerakan manusia | *Convolutional* LSTM dengan berbagai varian. | Dataset *jester* ConvLSTM biasa mendapatkan akurasi 96,11%, varian (1) 95,12%, varian (2) 94,18%, varian (3) 95,13%, dan varian (4) 95.10%. Sedangkan *Dataset IsoGD* ConvLSTM biasa 52,01%, varian (1) 55,98%, varian (2) 43,93%, varian (3) 53,27%, dan varian (4) 54,11%. |
| 2 | You, *et. al.* [5], *2019* | Deteksi kantuk berkendara secara *real-time* dengan mempertimbangkan perbedaan individu | DCCNN untuk ekstraksi fitur dan SVM untuk klasifikasi | Dari proses pelatihan dan evaluasi yang dilakukan didapatkan akurasi 94,80%. |
| 3 | Ye *et. al.* [14], *2019* | Pengenalan aktivitas manusia | *Two-Stream Conovlutional Network* dan *Convolutiona* | Menghasilkan Akurasi 69,4% untuk Dataset HMDB51 dan |

| | | | | |
|---|---|---|---|---|
| | | | *l* LSTM | 93,9% untuk Dataset UCF-101. |
| 4 | Duman dan Erdem [11], 2019 | Deteksi anomali pada video | Convolutional Autoencoder and *Convolutional* LSTM | Didapatkan AUC 89,5% untuk Dataset avenue, 92,4% untuk Ped1, dan 92,9% untuk Ped2 |
| 5 | Altameem, *et. al* [6], 2021 | Identifikasi dan deteksi dini kantuk pengemudi. | *Hybrid machine learning* dan SVM | Skenario pertama memiliki akurasi 91%, skenario kedua memiliki akurasi 81%, skenario ketiga menghasilkan akurasi 93%, dan skenario keempat menghasilkan akurasi 68%. |
| 6 | Sharma, *et. al.* [30], 2022 | Deteksi kelelahan dan sistem kewaspadaan kantuk secara *real time* | *Deep convolutional* dan SVM | Penelitian ini menghasilkan rata-rata akurasi 83,25% pada semua skenario. |
| 7 | Mousavikia *et. al.* [31], 2022 | Desain dan implementasi sistem deteksi kantuk pengemudi | CNN yang diimplementasikan pada *processor* RiscV yang dimodifikasi *Field-Programmabl* | Model dapat mengklasifikasikan empat ekspresi utama pengemudi yaitu normal, gangguan, menguap, dan tidur. Akurasi yang diperoleh |

| | | | | |
|---|---|---|---|---|
| | | | *e Gate Array* (FPGA). | adalah 81,07% |
| 8 | Yılmaz dan Akcayol [32], 2022 | Deteksi kantuk pengemudi | AlexNet, VGG16, VGG19 dan VGGFaceNet + LSTM | VGG19+LSTM menghasilkan akurasi 90,53%, VGG16+LSTM 89,39%, AlexNet+LSTM 63,91%, dan VGGFaceNet+LSTM 84,94%. |
| 9 | Vijaypriya dan Uma [33], 2023 | Deteksi kantuk berbasis fitur wajah | *Multi-Scale Convolutional Neural Network.* (MCNN) dan *Flamingo Search Algorithm* (FSA). | Jika pengemudi menguap atau mata tertutup maka diklasifikasikan sebagai mengantuk, sedangkan jika Mata terbuka, mulut tertutup, atau berbicara diklasifikasikan sebagai tidak mengantuk. Dari hasil evaluasi didapatkan akurasi 98,38% untuk *Dataset* YAWDD dan 98,26% untuk NTHU-DDD. |
| 10 | Abdullah *et. al.* [19], 2023 | Deteksi kelelahan pengemudi | *Raspberry-Pi* dan klasifikasi | Metode *deep learning* menghasilkan rata- |

| | | | | |
|---|---|---|---|---|
| | | | menggunakan *Deep Learning*, ANN, dan EAR (*Eye Aspect Ratio*). | rata akurasi 73,67%, metode ANN 88%, dan EAR menghasilkan akurasi 90% dan 100%. |
| 11 | Kim *et. al*. [27], 2023 | Identifikasi action dalam game | *Depthwise Separable Convolution* dan LSTM | Penelitian ini menghasilkan macro F1-Score 0,4475. |
| 12 | Mobsite *et. al*. [15], 2023 | Klasifikasi aktivitas manusia dan deteksi jatuh | GHM untuk ektstraksi fitur, *Dilated Convolutional* LSTM dan LDNN untuk pelatihan, dan *Depthwise Separable Convolution* untuk reduksi parameter pada LDNN | Menghasilkan F1-Score 98,46% untuk deteksi aktivitas, dan 98,87% untuk deteksi jatuh |

Dari beberapa penelitian yang telah dilakukan, banyak penelitian tentang deteksi kantuk atau kelelahan pengemudi dengan menggunakan data video wajah dari pengemudi. Akan tetapi mayoritas penelitian tersebut tidak memperhatikan urutan waktu dari data yang digunakan, sedangkan data video merupakan data yang juga mengandung fitur *temporal.* Metode yang digunakan pada beberapa penelitian tersebut juga bukan merupakan metode yang secara khusus digunakan untuk menangani data urutan waktu.

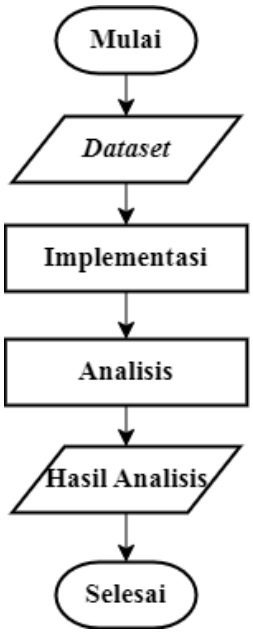
Metode yang dinilai dapat menangani data urutan waktu yang berupa video dengan baik salah satunya yaitu *Convolutional* LSTM. Dari beberapa penelitian yang menggunakan *Convolutional* LSTM untuk data video menunjukkan performa yang bagus, akan tetapi metode tersebut belum banyak digunakan untuk kasus deteksi kantuk pengemudi.

Selain itu, pada *Convolutional* LSTM memiliki parameter yang relatif besar yang berakibat pada waktu komputasi untuk pelatihan dan prediksi. *Depthwise Separable Convolution* terbukti secara efektif mengurangi jumlah parameter pada daripada operasi konvolusi biasanya. Dengan demikian Depthwise Separable Convolution yang diterapkan untuk menggantikan operasi konvolusi pada *Convolutional* LSTM dinilai dapat mengurangi jumlah parameternya.

# BAB III
# METODE USULAN

## 3.1. Alur Penelitian

Berdasarkan permasalahan yang telah dikemukakan sebelumnya, Alur dari penelitian ini secara garis besar direpresentasikan dengan diagram pada Gambar 3. 1.




**Gambar 3. 1. Alur Penelitian**

## 3.2. *Dataset*

Pada penelitian ini *Dataset* yang digunakan merupakan data yang berupa kumpulan video wajah dari seseorang yang sedang mengemudi. Video tersebut diambil dengan posisi kamera berada pada *dashboard* kendaraan. Video diambil dengan pengemudi dalam dua kondisi yaitu pada saat pengemudi mengantuk dan tidak mengantuk.

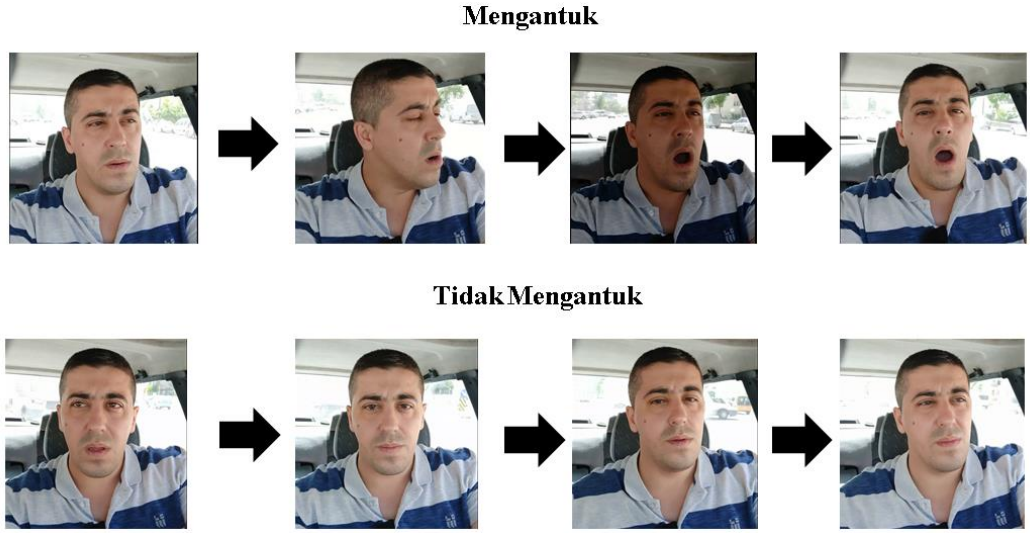
*Dataset* ini didapatkan dari *kaggle* dengan nama SUST-DDD atau *Sivas University of Science and Technology Driver Drowsiness Dataset*. *Dataset* ini diterbitkan pada tahun 2022 [32]. *Dataset* ini memiliki 2 kelas yaitu *drowsinesss* (mengantuk), dan *not drowsiness* (tidak mengantuk). *Dataset* ini berupa video yang memiliki durasi sekitar 10 detik. Jumlah video untuk kelas *drowsiness* yaitu 975 dan kelas *not drowsiness* yaitu 1099. Sehingga total data yang digunakan adalah 2074 video. Pada *Dataset* ini terdapat 19 pengemudi dengan rincian 3

perempuan dan 16 laki-laki. Peserta merekam video momen berkendara mereka dengan ponsel di kendaraan masing-masing pada waktu yang diinginkan. Selama perekaman, peserta tidak diminta untuk bertindak apa pun, dan reaksi serta gejala dicatat sesuai alur alaminya. Kelebihan SUST-DDD daripada *Driver Drowsiness Dataset* (DDD) lain yaitu SUST-DDD merupakan rekaman dari seseorang yang benar-benar sedang mengemudi di jalan raya, bukan sekedar simulasi berpura-pura mengantuk di lingkungan simulasi [32]. *Dataset* dapat di akses pada https://www.kaggle.com/datasets/esrakavalci/sust-ddd.

**Tabel 3. 1 Detail *Dataset* SUST-DDD**

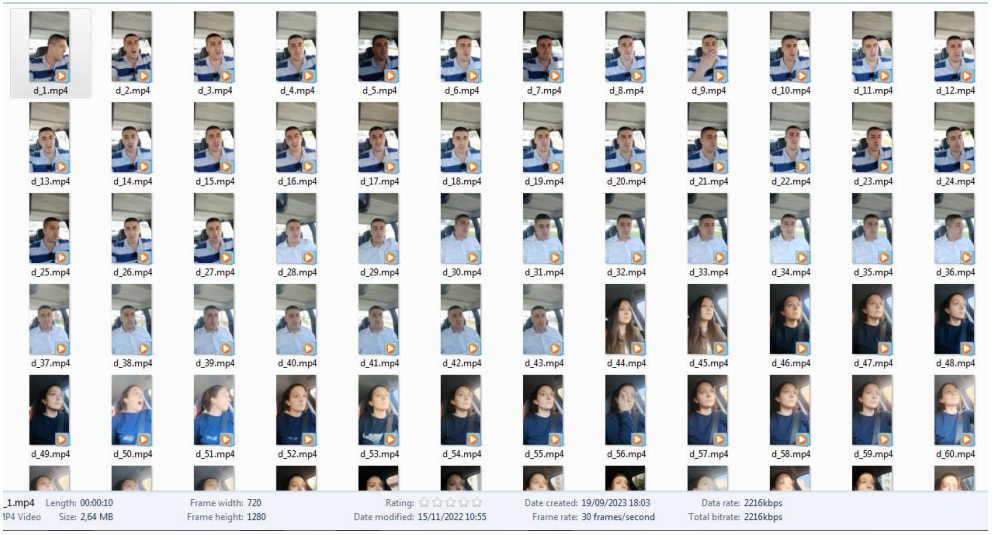
| Kelas | pria | wanita | Total |
|---|---|---|---|
| *Drowsiness* (Mengantuk) | 873 | 102 | 975 |
| *Not Drowsiness* (Tidak Mengantuk) | 961 | 138 | 1099 |

SUST-DDD merupakan *Dataset* yang berupa video, sehingga perlu adanya ekstraksi frame. Contoh sebuah video dari *Dataset* yang sudah dilakukan ekstraksi frame dapat diperhatikan pada Gambar 3.2.




**Gambar 3. 2. *Dataset***

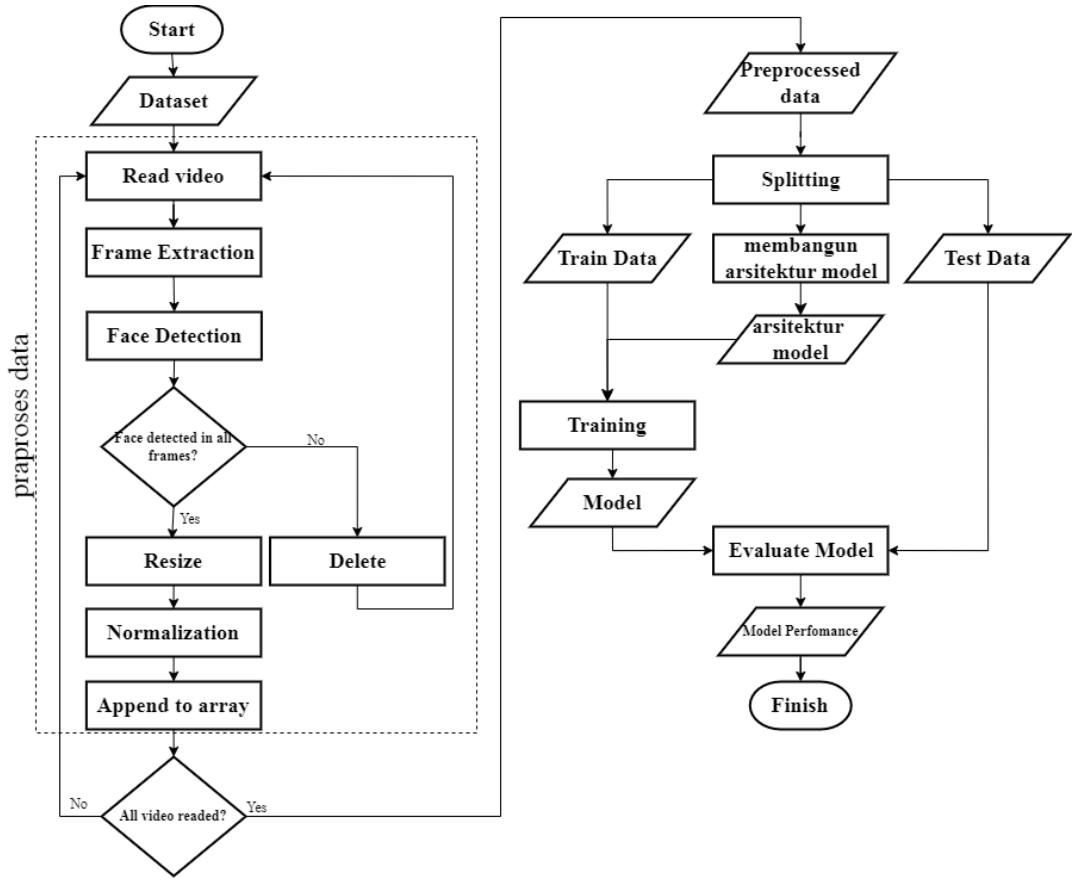
Dari Gambar 3. 2 dapat diperhatikan, perbedaan untuk kelas mengantuk dan tidak mengantuk yaitu pola perubahan wajah dari pengemudi. Untuk kelas mengantuk, pengemudi terlihat memiliki mata yang lebih redup dan menguap. *Dataset* yang digunakan dapat diperhatikan pada Gambar 3. 3.



**Gambar 3. 3. Folder Dataset**

## 3.3. Implementasi

Alur dari tahap implementasi yang akan dilakukan dapat diperhatikan pada diagram alir pada Gambar 3. 4.




**Gambar 3. 4. Diagram Alir Implementasi dan Uji Coba**

Dari diagram alir pada Gambar 3. 4 tahapan implementasi dan uji coba yang dilakukan adalah sebagai berikut:

### 3.3.1. Praproses Data

Dataset video yang akan digunakan dilakukan praproses terlebih dahulu. Tahap praproses meliputi:

1. *Frame Extraction*, *Frame Extraction* adalah sebuah proses dimana data video akan diubah menjadi kumpulan *frame* per *frame*. Dari sebuah video akan dijadikan 20 *frame* [32].
2. *Face Detection*, Setelah ekstraksi *frame*, setiap *frame* tersebut akan dilakukan deteksi wajah, dikarenakan untuk mengidentifikasi kantuk yang diperlukan hanyalah wajah dari pengemudi. Jika dari frame tersebut terdeteksi wajah maka akan dilakukan *cropping* atau pemotongan pada bagian wajah tersebut, sehingga selain dari wajah akan dibuang, dan yang digunakan adalah wajahnya saja. Jika dari sebuah video tidak terdeteksi wajah maka video tersebut tidak akan digunakan atau dibuang, sehingga data akan berkurang setelah dilakukan deteksi wajah. Perbandingan data sebelum dan setelah deteksi wajah dapat dilihat pada Gambar 3. 5.


**Gambar 3. 5. Data Sebelum dan Sesudah Deteksi Wajah**

3. *Resize*, hal ini dilakukan untuk memenuhi kebutuhan dari arsitektur, citra hasil ekstraksi *frame* dan deteksi wajah akan diubah ukurannya. Hal ini juga dapat mengurangi ukuran *feature map* agar model yang digunakan dapat membaca tingkat kemiripan yang tinggi dari setiap kelas.

4. *Normalization*, Normalisasi citra dengan membagi nilai setiap pikselnya dengan 255 digunakan untuk mengubah rentang nilai piksel pada citra menjadi 0 hingga 1. Hal ini dilakukan untuk memudahkan proses pengolahan citra selanjutnya. Normalisasi citra juga dapat membantu mengurangi efek cahaya dan meningkatkan kontras pada citra.

### 3.3.2. Splitting

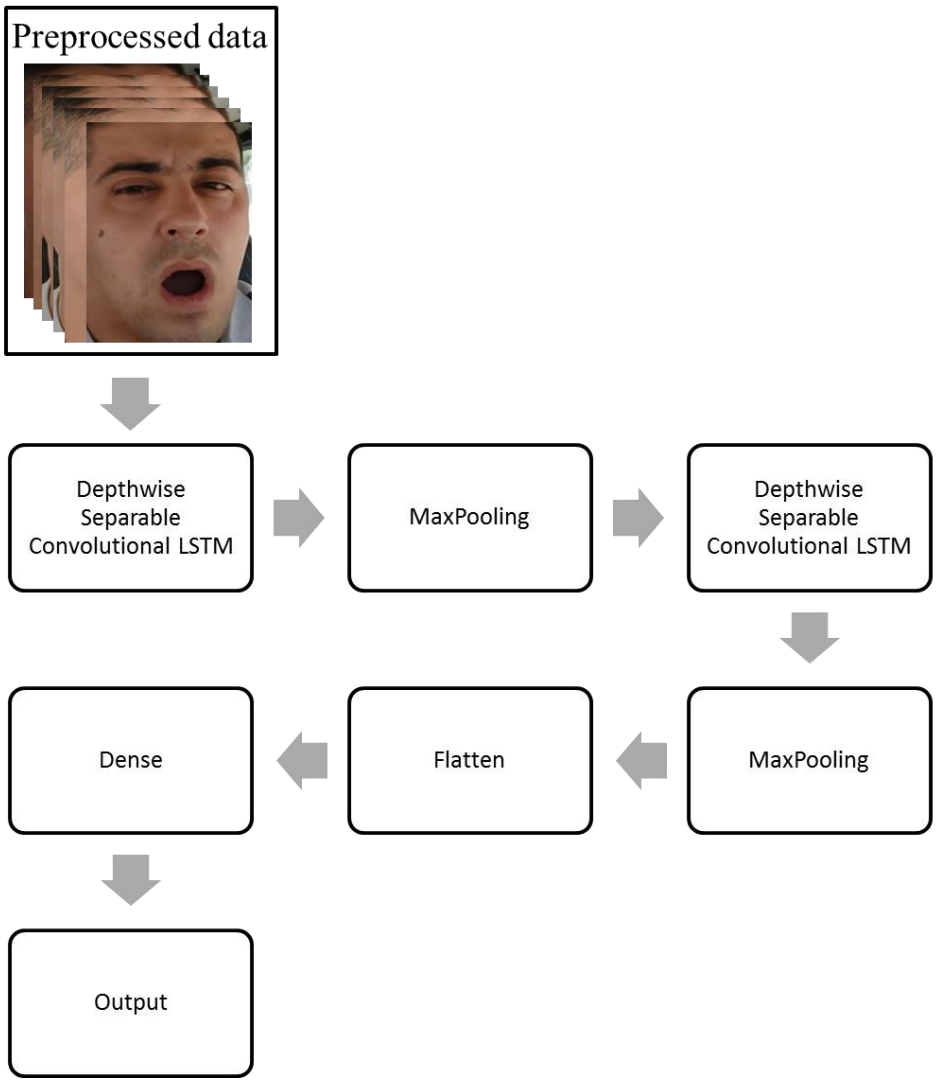
Splitting merupakan proses dimana dataset dilakukan pembagian data sesuai dengan kebutuhan. Splitting data akan menghasilkan kumpulan data *train* dan data *test*. Data *train* digunakan dalam proses *training* atau pelatihan, sedangkan data test digunakan untuk proses evaluasi. Dataset akan di-*splitting* dengan rasio 8: 2 [33] secara acak. Sehingga 80% data digunakan untuk pelatihan dan 20% data digunakan untuk evaluasi. Dari 80% data *training* dilakukan *splitting* lagi dengan rasio 8:2. *Splitting* ini menghasilkan data training dan data validasi. Kedua data ini nantinya digunakan dalam proses *training* . *Pada splitting* ini yang dibagi yaitu jumlah datanya, bukan *frame*-nya, sehingga 1 data tetap memiliki 20 *frame* tanpa *terpecah*.

### 3.3.3. Arsitektur Model

Untuk menangani data yang berhubungan dengan deret waktu atau *time series data* dibutuhkan metode yang dapat memproses *sequence*. Salah satu metode yang dapat memproses *sequence* yaitu *Recurrent Neural Network* atau RNN. Akan tetapi RNN memiliki masalah *input* yang diproses terlalu jauh dari masa lampau. Untuk mengatasi masalah tersebut dapat menggunakan LSTM. LSTM dinilai dapat mengatasi masalah pada korelasi urutan waktu baik dalam waktu yang singkat maupun lama [8]. Meskipun demikian, LSTM tidak memungkinkan untuk menerima *input* yang berupa citra secara langsung, sehingga arsitektur LSTM akan dikombinasikan dengan metode *convolutional* agar dapat menerima input berupa citra secara langsung. Arsitektur ini dinamakan *Convolutional* LSTM. Lalu *Depthwise Separable Convolution* digunakan untuk mereduksi parameter pada proses konvolusi dengan harapan dapat mengurangi beban pelatihan pada model.

Sistem yang akan dirancang dalam penelitian ini secara umum yaitu data setelah praproses akan dimasukkan ke dalam arsitektur model untuk dilakukan

pelatihan atau proses training untuk menghasilkan model terbaik untuk mendeteksi kantuk. Arsitektur model yang digunakan terdiri dari beberapa *layer Depthwise Separable Convolutional* LSTM *layer*, *maxpooling*, dan *Fully Connected Layer*. Diagram rancangan sistem secara umum dapat diperhatikan pada Gambar 3. 6.



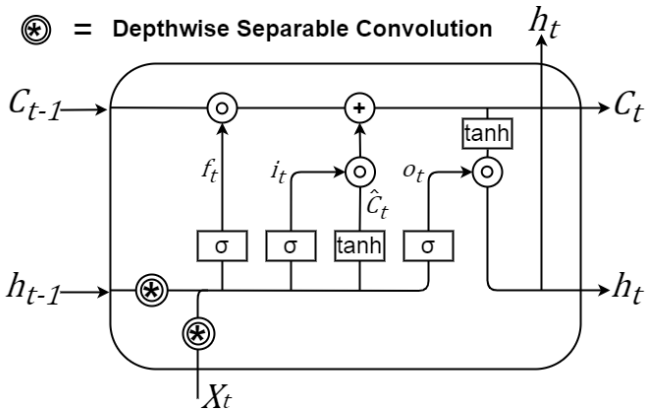

**Gambar 3. 6. Diagram Rancangan Sistem**

### 3.3.4. Depthwise Separable Convolutional LSTM

Arsitektur *Convolutional* LSTM yang dikombinasikan dengan *Depthwise Separable Convolution* dapat disebut juga dengan *Depthwise Separable Convolutional* LSTM. *Depthwise Separable Convolution* akan menggantikan operasi konvolusi pada arsitektur *Convolutional* LSTM. Meskipun demikian, *Depthwise Separable Convolution* tidak berhubungan langsung dengan fitur

temporal pada data. Depthwise Separable Convolution hanya digunakan untuk mendapatkan dan meneruskan fitur spatial dari data inputan [27].

Arsitektur Cell *Depthwise Separable Convolutional* LSTM dapat diperhatikan pada Gambar 3.7.




**Gambar 3. 7. *Depthwise Separable Convolutional* LSTM**

Dari diagram pada Gambar 3. 7 didapatkan persamaan dari *Depthwise Separable Convolutional* LSTM pada Persamaan (3.1) sampai dengan Persamaan (3.6).

$$i_t = \sigma(W_{hi} \circledast h_{t-1} + W_{xi} \circledast x_t + b_i) \quad (3.1)$$

$$f_t = \sigma(W_{hf} \circledast h_{t-1} + W_{xf} \circledast x_t + b_f) \quad (3.2)$$

$$\tilde{C}_t = tanh(W_{hc} \circledast h_{t-1} + W_{xc} \circledast x_t + b_c) \quad (3.3)$$

$$C_t = f_t \circ C_{t-1} + i_t \circ \tilde{C}_t \quad (3.4)$$

$$o_t = \sigma(W_{ho} \circledast h_{t-1} + W_{xo} \circledast x_t + b_o) \quad (3.5)$$

$$h_t = o_t \circ \tanh(C_t) \quad (3.6)$$

Dimana ⊛ merupakan operasi *Depthwise Separable Convolution.*

Jika dilihat pada operasi yang terdapat pada *Depthwise Separable Convolution*, Persamaan (4.1) sampai dengan Persamaan (4.6) dapat dijabarkan menjadi pada Persamaan (3.7) sampai dengan Persamaan (3.12).

$$i_t = \sigma(W_{hi}^{1\times1} * (W_{hi} \odot h_{t-1}) + W_{xi}^{1\times1} * (W_{xi} \odot x_t) + b_i) \quad (3.7)$$

$$f_t = \sigma(W_{hf}^{1\times1} * (W_{hf} \odot h_{t-1}) + W_{xf}^{1\times1} * (W_{xf} \odot x_t) + b_i) \quad (3.8)$$

$$\tilde{C}_t = tanh(W_{hc}^{1\times1} * (W_{hc} \odot h_{t-1}) + W_{xc}^{1\times1} * (W_{xc} \odot x_t) + b_i) \quad (3.9)$$

$$C_t = f_t \circ C_{t-1} + i_t \circ \tilde{C}_t \quad (3.10)$$

$$o_t = \sigma(W_{ho}^{1\times1} * (W_{ho} \odot h_{t-1}) + W_{xo}^{1\times1} * (W_{xo} \odot x_t) + b_i) \quad (3.11)$$

$$h_t = o_t \circ \tanh(C_t) \quad (3.12)$$

Dimana:

$\circledcirc$ : operasi *Depthwise Convolution*

$W_h$ : bobot atau kernel *Depthwise Convolution* dari *hidden*

$W_h^{1\times1}$ : bobot atau kernel *Pointwise Convolution* dari *hidden*

$W_x$ : bobot atau kernel *Depthwise Convolution* dari *input*

$W_x^{1\times1}$ : bobot atau kernel *Pointwise Convolution* dari input

Untuk menghitung parameter pada kombinasi *Depthwise Separable Convolution* dengan *Convolutional* LSTM memiliki cara yang berbeda dibandingkan dengan *Convolutional* LSTM pada umumnya. Untuk menghitung parameter pada metode ini dapat mengikuti langkah-langkah berikut

- Pada setiap *gate* menggunakan persamaan:

  $$W_{xDC} = 1 \times ukuran\ kernel \times channel\ citra\ input \quad (3.13)$$

  $$W_{xPC} = jumlah\ kernel \times 1 \times 1 \times channel\ citra\ input \quad (3.14)$$

  $$W_{hDC} = 1 \times ukuran\ kernel \times channel\ citra\ output \quad (3.15)$$

  $$W_{hPC} = jumlah\ kernel \times 1 \times 1 \times channel\ citra\ output \quad (3.16)$$

  Dimana:

  $W_{xDC}$ : Parameter *Depthwise Convolution* pada *input*

  $W_{xPC}$ : Parameter *Pointwise Convolution* pada *input*

  $W_{hDC}$ : Parameter *Depthwise Convolution* pada *hidden*

  $W_{hPC}$ : Parameter *Pointwise Convolution* pada *hidden*

- Pada *Convolutional* LSTM terdapat 4 *gate*, sehingga:

  $$Parameter\ DSCLSTM = 4 \times (W_{xDC} + W_{xPC} + W_{hDC} + W_{hPC}) \quad (3.17)$$

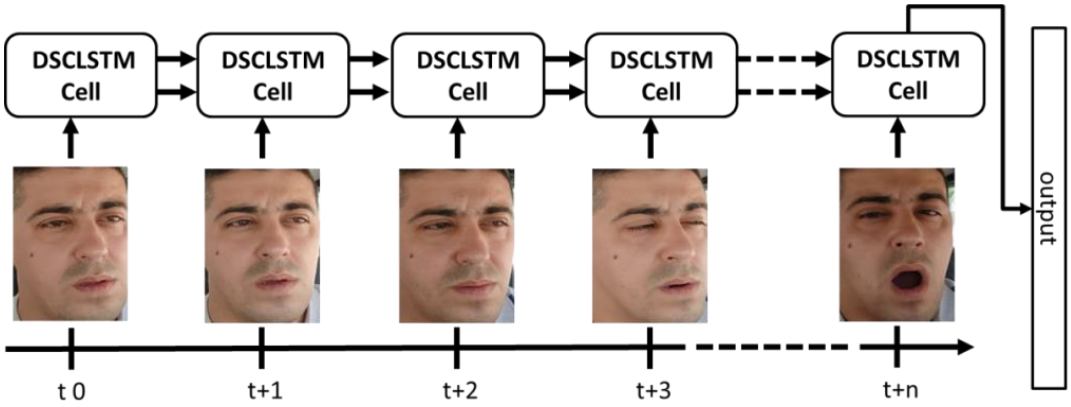
- Jika menggunakan bias, setiap gate ditambahkan dengan nilai bias. Sehingga persamaannya menjadi seperti pada persamaan 3.18.

  $$Parameter\ DSCLSTM = 4 \times (W_{xDC} + W_{xPC} + W_{hDC} + W_{hPC} + bias) \quad (3.18)$$

  *catatan: Nilai bias sama dengan nilai jumlah kernel yang digunakan.

Jumlah parameter pada suatu model yang menggunakan konsep *convolutional* tergantung jumlah kernel yang digunakan, ukuran kernel, dan juga ukuran citra *input*.

Diagram cara kerja *Depthwise Separable Convolutional* LSTM pada arsitektur model yang digunakan dapat diperhatikan pada Gambar 3. 8.




**Gambar 3. 8. Cara kerja *Depthwise Separable Convolutional* LSTM**

Pada Gambar 3. 8 setiap *frame* dari video akan dimasukkan ke dalam cell *Depthwise Separable Convolutional* LSTM yang nantinya akan diproses di dalamnya.

### 3.3.5. Pelatihan

*P*elatihan merupakan sebuah proses dimana arsitektur model yang telah dibangun dilakukan pelatihan dengan menggunakan data *training*. Proses pelatihan diharapkan mampu melatih model agar dapat melakukan sesuatu seperti yang diharapkan. Proses pelatihan dilakukan dengan menggunakan beberapa skenario uji coba.

Skenario uji coba dilakukan dengan tujuan untuk menjawab pertanyaan dari penelitian yang dilakukan. Untuk mengetahui permasalahan dari performa model dilakukan proses pelatihan menggunakan arsitektur dasar menggunakan *Depthwise Separable Convolutional* LSTM. Pelatihan dilakukan *batch size* yang 32, dan menggunakan fungsi Adam sebagai optimizernya [15]. Sedangkan untuk mengetahui pengaruh penggunaan *Depthwise Separable Convolution* terhadap efisiensi dan efektifitas model, dilakukan juga proses pelatihan tanpa menerapkan *Depthwise Separable Convolution* atau hanya dengan menggunakan *Convolutional* LSTM dengan *hyperparameter* yang sama. *Hyperparameter* yang digunakan yaitu Epoch 10 [34], 50 [5], dan 100 [33]. Setiap Epoch menggunakan learning rate 0.001 [34], 0.0001, dan 0,00001 [34]. Rincian skenario uji coba dapat diperhatikan pada Tabel 3. 2.

**Tabel 3. 2 Skenario Uji Coba**

| **Arsitektur** | **Nilai *Hyperparameter*** | |
|---|---|---|
| | ***Learning rate*** | ***Epoch*** |
| *Depthwise Separable Convolutional* LSTM | 0,001 [34] | 10 [34] |
| | 0,0001 | 50 [5] |
| | 0,00001 [34] | 100 [33] |
| *Convolutioal* LSTM | 0,001 | 10 |
| | 0,0001 | 50 |
| | 0,00001 | 100 |

### 3.3.6. Evaluasi

Model yang dihasilkan dari proses *training* akan dilakukan evaluasi untuk mengetahui performa dari model yang telah dilatih. Proses evaluasi dilakukan dengan data *testing* untuk menghasilkan *confusion matrix*. Dari *confusion matrix* yang diperoleh dapat dihitung nilai F1-*score*.

## 3.4. Analisis

Proses analisis dilakukan dengan menganalisis jumlah parameter yang digunakan pada setiap metode. Selain itu analisis juga dilakukan pada hasil pelatihan dan evaluasi yang telah dilakukan. Hasil pelatihan dan evaluasi yang akan dianalisis merupakan waktu pelatihan, waktu prediksi, dan nilai F1-*score*. Pada proses ini akan dilakukan membandingkan jumlah parameter yang digunakan pada kedua metode untuk diketahui persentase penurunan parameter jika menggunakan metode yang diusulkan (*Depthwise Separable Convolutional* LSTM). Selain itu, data waktu pelatihan, waktu prediksi, dan nilai F1-*score* untuk setiap metode akan dibandingkan untuk mengetahui seberapa besar pengaruh dari penurunan parameter yang terjadi.

# BAB IV
# HASIL DAN PEMBAHASAN

## 4.1. Lingkungan Uji Coba

Untuk melaksanakan uji coba dibutuhkan perangkat keras dan juga beberapa perangkat lunak. Perangkat keras yang digunakan untuk penelitian ini merupakan laptop Toshiba Satellite P745, sedangkan perangkat lunak yang digunakan merupakan *Google Drive* dan *Google Colaboratory Notebooks*. Pada penelitian ini dibutuhkan *Google Drive* dengan Penyimpanan Ekstra dan *Google Colaboratory Notebooks* dengan layanan *Pro* untuk mendukung penelitian. *Google Drive* dengan penyimpanan ekstra dibutuhkan untuk menyimpan *Dataset*, data hasil ekstraksi *frame*, dan juga model yang telah dilatih. *Google Colaboratory Notebooks* digunakan untuk menjalankan code program untuk uji coba atau penelitian. Layanan pro dibutuhkan karena program membutuhkan RAM besar. Spesifikasi perangkat yang digunakan dalam penelitian ini dapat diperhatikan pada Tabel 4. 1 hingga Tabel 4. 3.

**Tabel 4. 1. Spesifikasi Perangkat Keras**

| No | Kebutuhan | Jenis |
|---|---|---|
| 1 | Sistem Operasi | Windows 7 Ultimate |
| 2 | CPU | Intel(R) Core(TM) i5-2410M CPU @ 2.30GHz |
| 3 | GPU | NVIDIA GeForce 525M |
| 4 | RAM | 8 GB |
| 5 | Bahasa Pemrograman | Python |

**Tabel 4. 2. Spesifikasi Colaboratory Notebooks**

| No | Kebutuhan | Jenis |
|---|---|---|
| 1 | Layanan | Pro |
| 2 | RAM | 51 GB |
| 3 | CPU | Intel(R) Xeon(R) CPU @ 2.20GHz |

**Tabel 4. 3. Spesifikasi perangkat lunak**

| No | Kebutuhan | Versi | Fungsi |
|---|---|---|---|
| 1 | numpy | 1.25.2 | Melakukan manipulasi matriks dan operasi matematika |

| | | | |
|---|---|---|---|
| 2 | matplotlib | 3.7.1 | Melakukan visualisasi |
| 3 | OpenCV | 4.8.0 | Melakuan proses manipulasi citra mulai dari membaca, mengubah, dan pra-proses lainnya |
| 4 | scikit-learn | 1.2.2 | Digunakan untuk pembagian data serta mendapatkan nilai evaluasi model berupa *confusion matrix* dan F1-Score |
| 5 | tensorflow | 2.15.0 | Digunakan untuk membangun arsitektur model serta menyimpannya |
| 6 | Face-recognition | 1.3.0 | Digunakan untuk mengambil gambar wajah dari frame |
| 7 | SepConvLSTM | - | Digunakan untuk layer utama Depthwise Separable Convolutional LSTM. Library ini didapatkan dari https://github.com/zahid58/TwoStreamSepConvLSTM_ViolenceDetection/blob/master/sep_conv_rnn.py |

Untuk mempersiapkan lingkungan uji coba yang berupa software seperti yang dipaparkan pada Tabel 4. 3 menggunakan kode program pada Lampiran 1.1.

## 4.2. Implementasi dan Uji Coba

Tahapan penelitian dan uji coba terdiri dari beberapa tahapan yaitu pra-proses data, membagi dataset, membangun arsitektur model, menentukan data *train* dan data validasi, melakukan pelatihan, dan evaluasi model

### 4.2.1. Praproses dan Deteksi Wajah

Sebelum data digunakan untuk pelatihan, data harus diproses terlebih dahulu agar data digunakan untuk pelatihan menggunakan metode yang diinginkan. Pemrosesan data sebelum pelatihan ini biasa disebut *data preprocessing* atau praproses data. Pada penelitian ini data diletakkan pada google drive sehingga perlu dilakukan persiapan terlebih dahulu. Persiapan ini dilakukan menggunakan kode program pada Lampiran 1.2.

Pada praproses data, dataset yang berupa video diubah menjadi kumpulan *frame* per *frame* citra. Setiap video diubah menjadi 20 *frame*. Pada setiap *frame*

akan dilakukan deteksi wajah dengan menggunakan *library* face_recognition dari python. Penggunaan *library* ini mudah dilakukan karena tidak memerlukan banyak sumber daya dan tidak perlu mengatur banyak parameter. Pada proses ini, jika wajah terdeteksi dari suatu *frame,* maka akan dilakukan *cropping* atau pemotongan pada wajah tersebut, sehingga yang digunakan merupakan gambar wajahnya saja. Jika dari sebuah *frame* tidak terdeteksi wajah, maka video tersebut tidak akan digunakan atau dibuang, sehingga data akan berkurang setelah dilakukan deteksi wajah. Proses deteksi wajah dapat diperhatikan pada Gambar 3. 5.

Dari total total 2074 data, setelah dilakukan deteksi dan *croping* pada bagian wajah dengan menggunakan *library face_recognition*, data berkurang menjadi 1408 data. Sehingga terdapat 666 data yang tidak terdeteksi wajahnya. Setelah citra wajah didapatkan selanjutnya yaitu merubah ukurannya menjadi 100×100 pixel. Hal ini dilakukan untuk mengurangi ukuran citra asli agar lebih mudah dilakukan pelatihan dan tidak memberatkan sistem. Setelah dirubah ukurannya, citra dilakukan normalisasi dengan membagi nilai setiap pixelnya dengan 255. Hal ini dilakukan agar lebih memudahkan proses pelatihan. Hasil Akhir dari tahap pra-proses yaitu *array* dengan ukuran (1408,20,100,100,3) dimana 1408 mewakili jumlah data, 20 merupakan jumlah *frame,* (100,100) merupakan lebar dan tinggi dari citra *frame,* dan 3 merupakan channel dari citra *frame.* Praproses data pada penelitian ini dilakukan dengan menggunakan kode program pada Lampiran 1.3.

### 4.2.2. Membagi dataset

Dari 1408 data yang ada akan dibagi menjadi data *train* untuk proses pelatihan dan data *test* untuk proses evaluasi. Pembagian data dilakukan dengan rasio 8:2 secara acak. Dengan rasio tersebut nantinya 80% data digunakan untuk proses pelatihan, dan 20% data digunakan untuk evaluasi. Sehingga 1126 data akan digunakan untuk pelatihan dan 282 data akan digunakan untuk evaluasi. Pembagian data ini yang dibagi yaitu jumlah datanya, bukan *frame*-nya, sehingga 1 data tetap memiliki 20 *frame* tanpa *terpecah*. Untuk membagi data menggunakan kode program pada Lampiran 0.

Pada pembagian data ini data diacak terlebih dahulu sebelum dilakukan pembagian, sehingga pada data pelatihan dan data evaluasi memiliki jumlah setiap kelas yang sama atau tidak jauh berbeda. Detail jumlah data dari awal proses sampai dengan proses membagi dataset dapat diperhatikan pada Tabel 4. 4.

**Tabel 4. 4. Perubahan Jumlah Data**

| | **Dataset** | **Setelah deteksi wajah** | **Setelah pembagian dataset** | |
|---|---|---|---|---|
| | | | **Train data** | **Test data** |
| Mengantuk | 975 | 708 | 563 | 145 |
| Tidak Mengantuk | 1099 | 700 | 563 | 137 |
| **Total** | **2074** | **1408** | **1126** | **282** |

Pada Tabel 4. 4 yang dicantumkan merupakan jumlah dari data. Setiap data tersebut terdiri dari 20 *frame* dengan ukuran 100×100 dan memiliki *channel* RGB

### 4.2.3. Membangun Arsitektur Model

Metode yang digunakan yaitu dengan menggunakan *Depthwise Separable Convolutional* LSTM. Pada metode ini *Depthwise Separable Convolution* digunakan untuk menggantikan operasi konvolusi pada setiap *gate Convolutional* LSTM. Arsitektur model yang dibangun terdiri dari beberapa *layer* yaitu *Depthwise Separable Convolutional* LSTM *layer*, *maxpooling layer*, *flatten*, dan *dense layer*. Rincian *layer* yang digunakan dapat diperhatikan pada Tabel 4. 5.

**Tabel 4. 5. Arsitektur model Depthwise Separable Convolutional LSTM dan parameternya**

| Layer | Attribut | Output Shape |
|---|---|---|
| Depthwise Separable Convolutonal LSTM 2D | filters = 32, kernel_size = (3, 3), padding=”same”, return_sequences=True, | (None, 20, 100, 100, 32) |
| Max Pooling 3D | pool_size=(1, 2, 2), padding='same' | (None, 20, 50, 50, 32) |
| Depthwise Separable Convolutonal LSTM 2D | filters = 64, kernel_size = (3, 3), padding=”same”, return_sequences=True, | (None, 20, 50, 50, 64) |
| Max Pooling 3D | pool_size=(1, 2, 2), padding='same' | (None, 20, 25, 25, 64) |
| Flatten | - | (None, 800000) |
| Dense | 2, Activation=”softmax” | (None, 2) |

Kode program untuk *layer Depthwise Separable Convolutonal* LSTM 2D tidak memanfaatkan *library Keras.* Kode program layer tersebut mengguankan kode program pada Lampiran 2.

Sebagai perbandingan, digunakan juga arsitektur model yang menggunakan *Convolutional* LSTM. Arsitektur model yang digunakan seperti pada Tabel 4. 5, akan tetapi pada layer *Depthwise Separable Convolutional* LSTM diganti dengan *Convolutional* LSTM. Rincian layer arsitektur model Convolutional LSTM yang akan digunakan dapat diperhatikan pada Tabel 4. 6.

**Tabel 4. 6. Arsitektur model *Convolutional* LSTM dan parameternya**

| Layer | Attribut | Output Shape |
|---|---|---|
| Convolutonal LSTM 2D | filters = 32,<br>kernel_size = (3, 3),<br>padding=”same”,<br>return_sequences=True, | (None, 20, 100, 100, 32) |
| Max Pooling 3D | pool_size=(1, 2, 2),<br>padding='same' | (None, 20, 50, 50, 32) |
| Convolutonal LSTM 2D | filters = 64,<br>kernel_size = (3, 3),<br>padding=”same”,<br>return_sequences=True, | (None, 20, 50, 50, 64) |
| Max Pooling 3D | pool_size=(1, 2, 2),<br>padding='same' | (None, 20, 25, 25, 64) |
| Flatten | - | (None, 800000) |
| Dense | 2,<br>Activation=”softmax” | (None, 2) |

Untuk membangun arsitektur model menggunakan kode program pada Lampiran 0.

### 4.2.4. Melakukan Pelatihan

Pelatihan merupakan sebuah proses dimana arsitektur model yang dibangun sebelumnya dilakukan pelatihan menggunakan Dataset pelatihan yang disiapkan sebelumnya Pada saat pelatihan, validasi yang digunakan adalah *holdout validation*, dimana dari 80% *Dataset* yang digunakan untuk pelatihan akan dibagi lagi dengan rasio 8:2 dimana 80% nya digunakan untuk untuk pelatihan itu sendiri dan 20% nya digunakan untuk validasi pada proses pelatihan. Pembagian data untuk validasi tidak dilakukan secara otomatis saat pelatihan bertujuan untuk data yang digunakan untuk pelatihan dan validasi untuk setiap skenario adalah data yang sama. Untuk membagi data tersebut memanfaatkan konsep pembagian data dari K-*fold Cross Validation*. Dikarenakan rasio yang

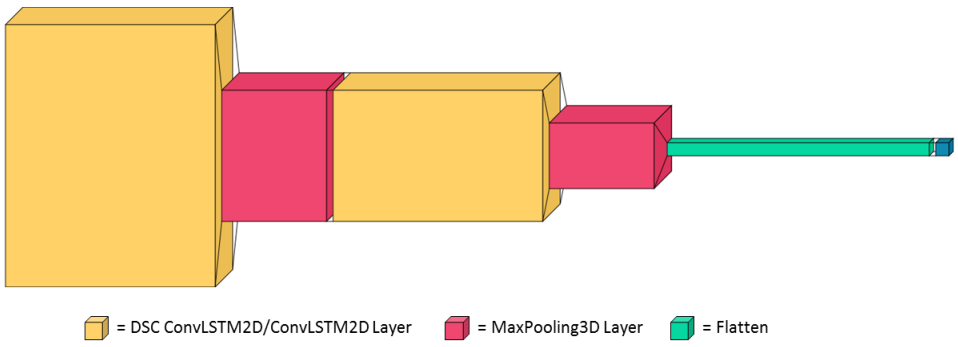
digunakan yaitu 8:2 sehingga nilai K yang digunakan yaitu 5. Dengan menggunakan nilai K sama dengan 5 akan diperoleh 5 subset data. Dari kelima *subset* tersebut dilakukan pemeriksaan untuk mengetahui subset mana yang memiliki persebaran setiap kelasnya seimbang. Sehingga *subset* tersebut yang akan digunakan untuk pelatihan. Dari pemeriksaan yang dilakukan, *subset* keempat yang memiliki persebaran setiap kelas yang merata. Sehingga yang digunakan untuk data validasi adalah *subset* keempat dan subset lainnya menjadi data *training*. Proses pemilihan subset yang akan digunakan untuk data validasi tersebut menggunakan kode program pada Lampiran 1.6. Sedangkan untuk Pelatihan dilakukan dengan menggunakan kode program pada Lampiran 0.

### 4.2.5. Evaluasi Model

Evaluasi model merupakan tahapan dimana model yang telah dilatih dilakukan uji coba prediksi dengan menggunakan data *testing* yang telah disiapkan sebelumnya. Evaluasi model bertujuan untuk mengukur seberapa jauh kemampuan model untuk memprediksi data diluar data untuk pelatihan dengan tepat. Metode pengukuran yang digunakan untuk evaluasi model adalah menggunakan *Confusion Matrix* yang nantinya digunakan untuk menghitung F1-*score*. Proses evaluasi dilakukan dengan menggunakan kode program pada Lampiran 1.8.

## 4.3. Pembahasan dan Analisis

Uji coba dilakukan dengan menggunakan metode kombinasi *Depthwise Separable Convolution* dengan *Convolutional LSTM* dengan menggunakan arsitektur model yang dijelaskan pada Tabel 4. 5. Atau Tabel 4. 6. Representasi dari arsitektur model yang digunakan untuk Uji coba dapat diperhatikan pada Gambar 4. 1.




**Gambar 4. 1. Representasi Model**

Jumlah kernel yang digunakan adalah 8 dengan ukuran (3×3). Pada proses pelatihan menggunakan *batch size* 32 dan *optimizer* adam. Sedangkan untuk *epoch* dan *learning rate* mengikuti skenario uji coba pada Tabel 3. 2.

### 4.3.1. Trainable Parameter

Pada arsitektur model yang digunakan untuk penelitian ini menggunakan 32 kernel *layer* pertama dan 64 kernel pada *layer* ketiga dengan ukuran kernel 3×3. Selain itu, data yang digunakan memiliki 3 channel yaitu RGB, sehingga jumlah parameter yang digunakan pada penelitian pada metode kombinasi *Depthwise Separable Convolution* dengan *Convolutional* LSTM pada layer pertama ini dapat dihitung dengan langkah-langkah berikut:

- Untuk menghitung parameter pada setiap gate dapat menggunakan Persamaan (3.13) sampai dengan Persamaan (3.16).

$$W_{xDC} = 1 \times 3 \times 3 \times 3$$
$$= 27$$
$$W_{xPC} = 32 \times 1 \times 1 \times 3$$
$$= 96$$
$$W_{hDC} = 1 \times 3 \times 3 \times 32$$
$$= 288$$
$$W_{hPC} = 32 \times 1 \times 1 \times 32$$
$$= 1024$$

- *Convolutional* LSTM memiliki 4 *gate* sehingga untuk menghitung jumlah total parameternya dapat menggunakan Persamaan (3.17). Tetapi karena terdapat bias, maka menggunakan Persamaan (3.18).

$$Parameter\ DSCLSTM = 4 \times (27 + 96 + 288 + 1024 + bias)$$
$$= 4 \times (27 + 96 + 288 + 1024 + 32)$$
$$= 5868$$

Sehingga total *trainable parameter* untuk layer *Depthwise Separable Convolutional* LSTM pada *layer* pertama yang digunakan pada penelitian ini yaitu 5868. Layer pertama ini memiliki ukuran *output* yaitu (*None*, 20, 100, 100, 32). *Output* ini nantinya digunakan untuk *input* pada *layer* berikutnya. *Layer* kedua merupakan *layer maxpooling3D* dengan ukuran *pool* (1, 2, 2) sehingga *output*-nya menjadi setengah dari *input*-nya. Data *input* memiliki ukuran (*None*, 20, 100,

100, 32) sehingga *output* dari *layer* kedua ini memiliki ukuran (*None*, 20, 50, 50, 32) dengan jumlah parameter yaitu 0. Parameter pada *layer* ketiga yang dihitung dengan menggunakan cara yang sama dengan layer pertama tetapi *channel input*-nya 32 (*channel* hasil dari *layer* sebelumnya) didapatkan jumlah parameter yaitu 28288 dengan ukuran *output* yaitu (*None*, 20, 50, 50, 64). *Layer* keempat sama dengan *layer* ketiga sehingga *output*-nya menjadi setengahnya yaitu yaitu (*None*, 20, 25, 25, 64) dengan jumlah parameter 0.

*Output* dari *layer* keempat dilakukan *flatten*, sehingga *output* pada tahap ini yaitu (*None*, 800000). Pada *layer* terakhir merupakan *layer dense* dengan jumlah 2 (sesuai jumlah kelas yang akan diklasifikasikan). Sehingga parameter pada *layer* ini yaitu :

$$\begin{aligned} \mathrm{P} &= 800000 \times 2 + 2 \\ &= 1600000 + 2 \\ &= 1600002 \end{aligned}$$

Sehingga Total parameter pada arsitektur model kombinasi *Depthwise Separable Convolution* dengan *Convolutional* LSTM yaitu :

$$\begin{aligned} \mathrm{P} &= 5868 + 0 + 28288 + 0 + 0 + 0 + 1600002 \\ &= 1634158 \end{aligned}$$

Ringkasan *layer*, ukuran *output*, beserta parameternya dapat diperhatikan pada Tabel 4. 7.

**Tabel 4. 7. Layer Arsitektur Model Depthwise Separable Convolutional LSTM Beserta Output dan jumlah parameternya**

| Layer | Output Shape | Parameter |
|---|---|---|
| Depthwise Separable Convolutional LSTM 2D | (None, 20, 100, 100, 32) | 5868 |
| Max Pooling 3D | (None, 20, 50, 50, 32) | 0 |
| Depthwise Separable Convolutional LSTM 2D | (None, 20, 50, 50, 64) | 28288 |
| Max Pooling 3D | (None, 20, 25, 25, 64) | 0 |
| Flatten | (None, 800000) | 0 |
| Dense | (None, 2) | 1600002 |

Pada arsitektur model yang tidak menggunakan *Depthwise Separable Convolution* memiliki arsitektur yang sama dengan arsitektur model yang menggunakan *Depthwise Separable Convolution*. Pada *layer Convolutional*

LSTM , jumlah parameter dapat dihitung dengan menggunakan Persamaan (2.22). Berikut perhitungan parameter pada *layer* pertama.

$$Parameter\ ConvLSTM = [3 \times 3 \times 32 \times (3 + 32) + 32] \times 4$$
$$= 40448$$

Layer ketiga memiliki jumlah kernel 64, sehingga jumlah paramternya yaitu:

$$Parameter\ ConvLSTM = [3 \times 3 \times 64 \times (32 + 64) + 64] \times 4$$
$$= 221440$$

Sehingga Total parameter pada arsitektur model Convolutional LSTM tanpa *Depthwise Separable Convolution* yaitu :

$$P = 40448 + 0 + 221440 + 0 + 0 + 0 + 1600002$$
$$= 1861890$$

Ringkasan *layer*, ukuran *output*, beserta parameternya dapat diperhatikan pada Tabel 4. 8.

**Tabel 4. 8. Layer Arsitektur Model Convolutional LSTM Beserta Output dan Jumlah Parameternya**

| Layer | Output Shape | Parameter |
|---|---|---|
| Convolutional LSTM 2D | (None, 20, 100, 100, 32) | 40448 |
| Max Pooling 3D | (None, 20, 50, 50, 32) | 0 |
| Convolutional LSTM 2D | (None, 20, 50, 50, 64) | 221440 |
| Max Pooling 3D | (None, 20, 25, 25, 64) | 0 |
| Flatten | (None, 800000) | 0 |
| Dense | (None, 2) | 1600002 |

Total parameter pada metode kombinasi Depthwise Separable Convolution yang digunakan pada penelitian ini yaitu 1634158, sedangkan pada metode Convolutional LSTM yaitu 1861890. Dengan menggunakan Persamaan (2.27) dapat didapatkan persentase penurunan parameter dari kedua arsitektur yang digunakan pada penelitian ini adalah sebesar 12.23% dengan jumlah layer dan atribut yang sama.

### 4.3.2. Hasil Evaluasi

Setelah dilakukan pelatihan dengan skenario uji coba yang ditentukan sebelumnya menghasilkan 18 model yang berbeda, dimana 9 model merupakan model yang dilatih dengan metode kombinasi Depthwise Separable Convolution dengan Convolutional LSTM dan 9 model lainnya merupakan model yang dilatih dengan metode Convolutional LSTM sebagai pembanding. Dari 18 model tersebut dilakukan evaluasi dengan menggunakan data *testing*. Evaluasi yang

digunakan pada penelitian ini menggunakan *Confusion Matriks* yang nantinya digunakan untuk menghitung F1-*score*. Selain itu dilakukan pencatatan waktu komputasi dari pelatihan dan juga untuk evaluasinya.

Dari hasil evaluasi Pada model kombinasi *Depthwise Separable Convolution* dengan *Convolutional* LSTM yang menggunakan *hyperparameter Epoch* 10 dan *learning rate* 0.001 didapatkan hasil prediksi:

```
array([1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 0, 1,
1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0,
0, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0,
0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0,
0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0,
1, 0, 0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1,
0, 0, 1, , 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1,
0, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1,
1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0])
```

Sedangkan label aslinya adalah

```
array([1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 0,
0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0,
0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1,
0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1,
1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1,
0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1,
1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0,
1, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1,
1, 0, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0])
```

Pada penelitian ini, yang dianggap kelas positif yaitu kelas 0, sehingga pada data tersebut jika hasil prediksi 0 dan kelas sebenarnya 0 maka dianggap *true positive*, jika hasil prediksi 0 dan kelas sebenarnya 1 maka dianggap *false positive*, jika hasil prediksi yaitu 1 dan kelas sebenarnya 1 maka dianggap *true negative*, sedangkan jika hasil prediksi 1 dan kelas sebenarnya 0 maka dianggap *false negative*. Dengan menggunakan teori tersebut, maka jumlah *true positive* yaitu 108, *false positive* 39, *true negative* 98, *false negative* 37.

Untuk menghitung akurasi dari model tersebut dapat dilakukan dengan menggunakan Persamaan (2.28).

$$Accuracy = \frac{108 + 98}{108 + 39 + 98 + 37}$$

$$Accuracy = \frac{206}{282}$$

$$Accuracy = 0.7305$$

Untuk menghitung nilai F1-*score*, dibutuhkan nilai *precision* dan *recall*. Untuk menghitung *precision* dapat dilakukan dengan menggunakan Persamaan (2.29).

$$Precision = \frac{108}{108 + 39}$$

$$Precision = \frac{108}{147}$$

$$Precision = 0.7347$$

Sedangkan untuk menghitung *recall* dapat menggunakan Persamaan (2.30).

$$Recall = \frac{108}{108 + 37}$$

$$Recall = \frac{108}{145}$$

$$Recall = 0.7448$$

Setelah nilai *precision* dan *recall* didapatkan, nilai F1-*score* dapat dihitung menggunakan Persamaan (2.31).

$$F1\ Score = 2 * (\frac{0.7448 * 0.7347}{0.7448 + 0.7347})$$

$$F1\ Score = 2 * (\frac{0.5472}{1.4795})$$

$$F1\ Score = 2 * (0.3699)$$

$$F1\ Score = 0.7398$$

Sehingga nilai F1-*Score* dari model yang dibangun dengan kombinasi *Depthwise Separable Convolution* dengan *Convolutional* LSTM dan dengan menggunakan *hyperparameter epoch 10* dan *learning rate 0.001* yaitu 0.7266.

Dengan menggunakan perhitungan tersebut hasil evaluasi dari semua model yang dihasilkan didapatkan data-data pada Tabel 4. 9 dan Tabel 4. ***10***.

**Tabel 4. 9. Data hasil pelatihan dan evaluasi metode *Depthwise Separable Convolutional* LSTM**

| hyperparameter | | Depthwise Separable Convolutional LSTM | | | | | |
|---|---|---|---|---|---|---|---|
| | | hasil | | | | | |
| epoch | Learning rate | waktu pelatihan (detik) | accuracy (%) | precision (%) | recall (%) | f1-score (%) | waktu prediksi (detik) |
| 10 | 0.001 | 4967 | 73.05 | 73.47 | 74.48 | 73.98 | 265 |
| | 0.0001 | 5424 | 73.05 | 80.53 | 62.76 | 70.54 | 267 |
| | 0.00001 | 5256 | 63.83 | 65.47 | 62.76 | 64.08 | 264 |
| rata-rata | | **5215.68** | **69.98** | **73.15** | **66.67** | **69.53** | **265.33** |
| 50 | 0.001 | 53047 | 72.7 | 73.61 | 73.1 | 73.36 | 264 |
| | 0.0001 | 28232 | 72.34 | 73.43 | 72.41 | 72.92 | 264 |
| | 0.00001 | 52852 | 67.02 | 67.33 | 69.66 | 68.47 | 205 |
| rata-rata | | **44710.33** | **70.69** | **71.46** | **0.71.72** | **71.58** | **244.33** |
| 100 | 0.001 | 51931 | 71.99 | 73.91 | 70.34 | 72.08 | 265 |
| | 0.0001 | 51683 | 70.92 | 71.14 | 73.1 | 72.11 | 266 |
| | 0.00001 | 52267 | 67.38 | 67.32 | 71.03 | 69.13 | 200 |
| rata-rata | | **51960.33** | **70.1** | **70.79** | **71.49** | **71.11** | **243.67** |

**Tabel 4. 10. Data hasil pelatihan dan evaluasi metode *Convolutional* LSTM**

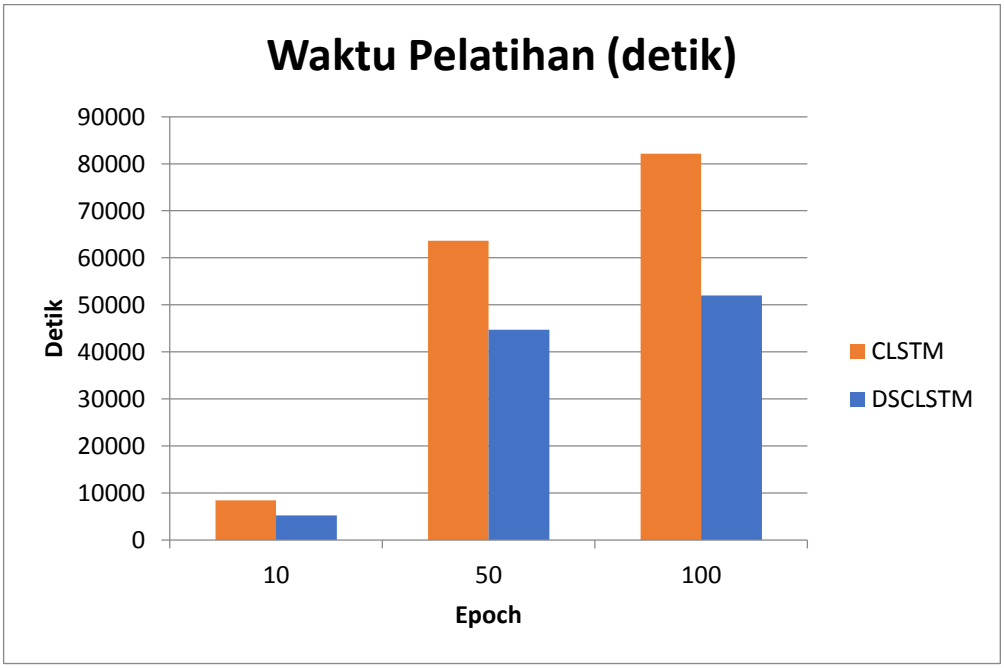
| hyperparameter | | Convolutional LSTM | | | | | |
|---|---|---|---|---|---|---|---|
| | | hasil | | | | | |
| epoch | Learning rate | waktu pelatihan (detik) | accuracy (%) | precision (%) | recall (%) | f1-score (%) | waktu prediksi (detik) |
| 10 | 0.001 | 8598 | 75.53 | 86.54 | 62.07 | 72.28 | 330 |
| | 0.0001 | 8170 | 73.4 | 72.15 | 78.62 | 75.25 | 386 |
| | 0.00001 | 8601 | 71.63 | 86.52 | 53.1 | 65.81 | 385 |
| rata-rata | | **8456.33** | **73.52** | **81.74** | **64.6** | **71.11** | **367** |
| 50 | 0.001 | 42992 | 73.76 | 77.52 | 68.97 | 72.99 | 384 |
| | 0.0001 | 77256 | 69.5 | 72.52 | 65.52 | 68.84 | 386 |
| | 0.00001 | 70675 | 73.04 | 72.54 | 76.55 | 74.5 | 388 |
| rata-rata | | **63641** | **72.1** | **74.19** | **70.35** | **72.11** | **386** |
| 100 | 0.001 | 81478 | 71.99 | 76.61 | 65.52 | 70.63 | 387 |
| | 0.0001 | 81291 | 74.82 | 77.61 | 71.72 | 74.55 | 343 |
| | 0.00001 | 83612 | 72.7 | 75.76 | 68.97 | 72.2 | 348 |
| rata-rata | | **82127** | **73.17** | **76.66** | **68.74** | **72.46** | **359.33** |

Tabel 4. 9 merupakan tabel data hasil pelatihan dan evaluasi model yang menggunakan metode kombinasi Depthwise Separable Convolution dan ConvLSTM sedangkan Tabel 4. 10 merupakan tabel data hasil pelatihan dan evaluasi model yang menggunakan metode Convolutional LSTM sebagai pembanding. Sedangkan waktu prediksi merupakan waktu saat model melakukan prediksi menggunakan data *testing*. Dari hasil evaluasi yang dilakukan didapatkan perbandingan rata-rata hasil pelatihan dan evaluasi pada setiap epoch yang tertera pada**Error! Not a valid bookmark self-reference.** sampai dengan Tabel 4. 13.

**Tabel 4. 11. Perbandingan Waktu Pelatihan setiap epoch**

| epoch | CLSTM (detik) | DSCLSTM (detik) | Persentase penurunan |
|---|---|---|---|
| 10 | 8456.33 | 5215.67 | 38.32% |
| 50 | 63641 | 44710.33 | 29.75% |
| 100 | 82127 | 51960.33 | 36.73% |

Grafik Perbandingan waktu pelatihan dapat dilihat pada Gambar 4. 2.

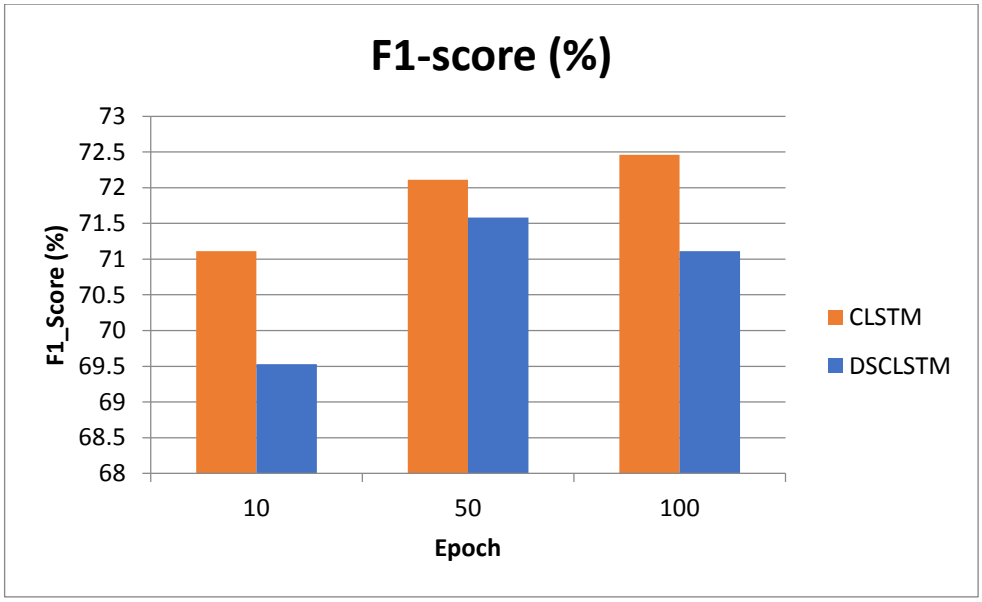



**Gambar 4. 2. Grafik Perbandingan Waktu Pelatihan**

**Tabel 4. 12. Perbandingan F1-Score setiap epoch**

| epoch | CLSTM (%) | DSCLSTM (%) | Persentase penurunan |
|---|---|---|---|
| 10 | 71.11 | 69.53 | 2.22% |
| 50 | 72.11 | 71.58 | 0.73% |
| 100 | 72.46 | 71.11 | 1.87% |

Grafik Perbandingan waktu F1-*score* dapat dilihat pada Gambar 4. 3.

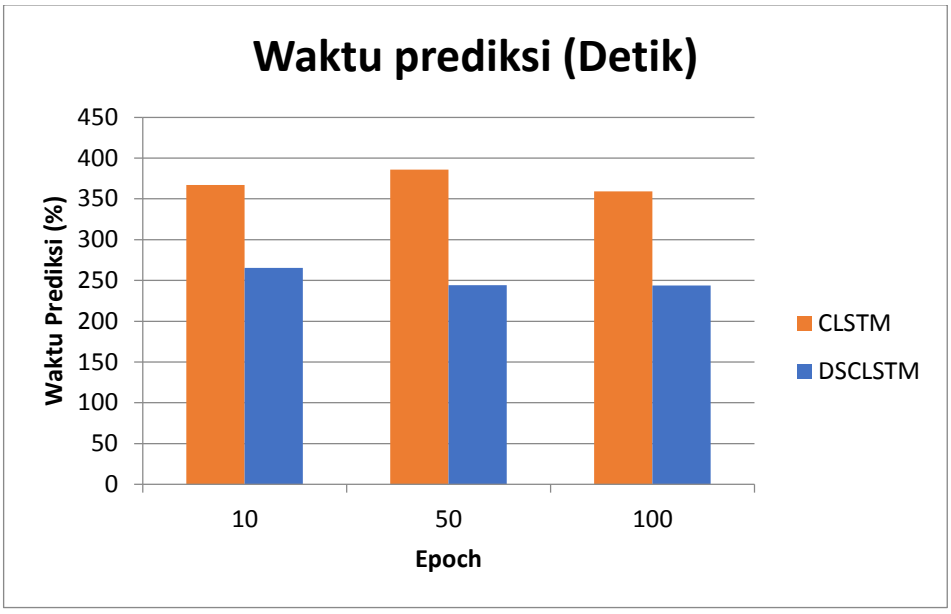



**Gambar 4. 3. Grafik Perbandingan F1-score**

**Tabel 4. 13. Perbandingan waktu prediksi epoch**

| epoch | CLSTM (detik) | DSCLSTM (detik) | Persentase penurunan |
|---|---|---|---|
| 10 | 367 | 265.33 | 27.70% |
| 50 | 386 | 244.33 | 36.70% |
| 100 | 359.33 | 243.67 | 32.19% |

Grafik Perbandingan waktu prediksi dapat dilihat pada Gambar 4. 4.




**Gambar 4. 4. Grafik Perbandingan Waktu Prediksi**

Pada Tabel 4. 11 sampai dengan Tabel 4. 13, diketahui dengan penurunan parameter dari metode yang digunakan, dapat memengaruhi waktu pelatihan, hasil f1-score dan waktu prediksi.

Secara keseluruhan dari skenario uji coba yang telah dilakukan, perbandingan rata-rata waktu pelatihan, f1-score, dan waktu prediksi mengalami penurunan yang dicantumkan pada Tabel 4. 14.

**Tabel 4. 14. perbandingan rata-rata waktu pelatihan, f1-score, dan waktu prediksi secara keseluruhan**

| | CLSTM | DSCLSTM | Persentase Penurunan |
|---|---|---|---|
| waktu pelatihan (detik) | 51408.11 | 33962.11 | 33.94% |
| f1-score (%) | 71.90 | 70.74 | 1.61% |
| waktu prediksi (detik) | 370.44 | 251.11 | 32.21% |

Jumlah parameter pada sebuah arsitektur model dinilai memiliki pengaruh terhadap akurasi dan waktu komputasi[18]. Parameter pada sebuah arsitektur model akan berpengaruh pada bobot dari sebuah model. Sehingga semakin banyak parameter, semakin banyak juga acuan yang akan digunakan untuk menentukan output dari suatu data. Dengan acuan yang banyak, model akan dapat menentukan output dengan lebih akurat. Akan tetapi, parameter juga berpengaruh pada banyaknya jumlah komputasi yang dilakukan pada sebuah model. Semakin banyak parameter, semakin banyak juga jumlah komputasi yang akan dilakukan pada sebuah model. Hal tersebut akan membuat model memiliki waktu yang lebih lama untuk pelatihan dan untuk prediksi. Sehingga, semakin banyak jumlah parameter model akan lebih akurat, akan tetapi waktu komputasi yang dibutuhkan akan lebih lama. Oleh karena itu dengan jumlah parameter yang lebih sedikit, model memiliki waktu komputasi yang lebih singkat, tetapi model memiliki akurasi yang lebih rendah meskipun tidak terlalu signifikan.

# BAB V
# PENUTUP

## 5.1. Kesimpulan

Berdasarkan proses penelitian dan uji coba yang telah dilaksanakan dengan menggunakan beberapa skenario pengujian, penelitian menghasilkan kesimpulan yaitu:

1. Model yang dibangun dengan menggunakan metode *Convolutional* LSTM yang menggunakan Depthwise Separable Convolution maupun tidak memiliki performa yang cukup baik untuk mengenali kantuk pengemudi dari data video. Hal tersebut dibuktikan dengan hasil evaluasi yang memiliki rata-rata f1-score sebesar 0,7 untuk data testing dan 0,5 untuk data baru. Meskipun demikian, model belum dapat dikatakan memiliki performa terbaik dikarenakan arsitektur model yang dibangun sangat sederhana dikarenakan keterbatasan sumber daya.
2. *Depthwise Separable Convolution* yang menggantikan operasi konvolusi pada *Convolutional* LSTM dapat mengurangi jumlah parameter secara signifikan pada metode *Convolutional* LSTM. Pada arsitektur model yang digunakan, arsitektur model *Depthwise Separable Convolutional LSTM* memiliki jumlah parameter 12.23% lebih sedikit dibandingkan dengan arsitektur model *Convolutional* LSTM. Pada saat pelatihan menggunakan seluruh skenario pengujian yang telah ditentukan, dengan penurunan parameter sejumlah 12.23%, proses pelatihan secara keseluruhan mengalami penurunan waktu sebesar 33.93%. Pada proses evaluasi, rata-rata dari keseluruhan f1-*score* mengalami penurunan sebesar 1.60%. Selain itu, rata-rata dari keseluruhan waktu prediksi pada saat evaluasi mengalami penurunan sebesar 32.21%.

Dari beberapa uraian tersebut, dapat disimpulkan bahwa *Convolutional* LSTM dapat dengan cukup baik mengenali kantuk pengemudi dari data video atau data yang memiliki fitur *spatio-temporal.* Sedangkan implementasi *Depthwise Separable Convolution* untuk menggantikan operasi konvolusi pada setiap *gate Convolutional* LSTM dapat secara efektif mengurangi jumlah parameter pada metode *Convolutional* LSTM yang mengakibatkan penurunan

waktu komputasi untuk pelatihan dan prediksi tanpa mengurangi performanya secara signifikan.

## 5.2. Saran

Berdasarkan pada hasil penelitian dan kesimpulan yang ada, penelitian yang telah dilakukan belum dapat dikatakan optimal dikarenakan beberapa kondisi. Sehingga saran untuk penelitian selanjutnya yaitu:

1. Penelitian dapat dilakukan dengan menggunakan arsitektur model yang lebih kompleks. Dengan menggunakan arsitektur yang lebih kompleks diharapkan model memiliki performa yang lebih baik untuk mengenali kantuk pengemudi dari data video.
2. Melakukan uji coba dengan menggunakan jumlah kernel yang berbeda. Dengan melakukan uji coba dengan lebih banyak kernel yang berbeda diharapkan mampu memberikan persentase penurunan waktu komputasi dan performanya dengan lebih akurat.
3. Melakukan penelitian dengan sumber daya yang lebih besar dan dengan menggunakan *processor* yang lebih canggih. Hal tersebut sangat dibutuhkan untuk melakukan penelitian dengan menggunakan arsitektur yang lebih kompleks

# DAFTAR PUSTAKA


[1] S. S. Jasim dan A. K. A. Hassan, "Modern drowsiness detection techniques: a review," *International Journal of Electrical and Computer Engineering*, vol. 12, no. 3, pp. 2986–2995, 2022, doi: 10.11591/ijece.v12i3.pp2986-2995.

[2] K. Gopalakrishna dan S. A. Hariprasad, "Real-time fatigue analysis of driver through iris recognition," *International Journal of Electrical and Computer Engineering*, vol. 7, no. 6, pp. 3306–3312, 2017, doi: 10.11591/ijece.v7i6.pp3306-3312.

[3] S. S. Jasim dan A. K. A. Hassan, "Driving sleepiness detection using electrooculogram analysis dan grey wolf optimizer," *International Journal of Electrical and Computer Engineering*, vol. 12, no. 6, pp. 6034–6044, 2022, doi: 10.11591/ijece.v12i6.pp6034-6044.

[4] F. N. Fajri, A. Tholib, dan W. Yuliana, "Application of Machine Learning Algorithm for Determining Elective Courses in Informatics Study Program," *Jurnal Teknik Informatika dan Sistem Informasi*, vol. 8, no. 3, pp. 485–496, 2022, doi: 10.28932/jutisi.v8i3.3990.

[5] F. You, X. Li, Y. Gong, H. Wang, dan H. Li, "A Real-time Driving Drowsiness Detection Algorithm with Individual Differences Consideration," *Institute of Electrical and Electronics Engineers Access*, vol. 7, pp. 179396–179408, 2019, doi: 10.1109/ACCESS.2019.2958667.

[6] A. Altameem, A. Kumar, R. C. Poonia, S. Kumar, dan A. K. J. Saudagar, "Early Identification dan Detection of Driver Drowsiness by Hybrid Machine Learning," *Institute of Electrical and Electronics Engineers Access*, vol. 9, pp. 162805–162819, 2021, doi: 10.1109/ACCESS.2021.3131601.

[7] X. Wei, Y. Liu, S. Gao, X. Wang, dan H. Yue, "An RNN-Based Delay-Guaranteed Monitoring Framework in Underwater Wireless Sensor Networks," *Institute of Electrical and Electronics Engineers Access*, vol. 7,

pp. 25959–25971, 2019, doi: 10.1109/ACCESS.2019.2899916.

[8] E. Ismanto dan N. Effendi, "An LSTM-based prediction model for gradient-descending optimization in virtual learning environments," *Computer Science and Information Technologies.*, vol. 4, no. 3, pp. 199–207, 2023, doi: 10.11591/csit.v4i3.p199-207.

[9] Z. Chao, F. Pu, Y. Yin, B. Han, dan X. Chen, "Research on real-time local rainfall prediction based on MEMS sensors," *Journal of Sensors*, vol. 2018, pp. 1–9, 2018, doi: 10.1155/2018/6184713.

[10] M. M. B. Ismail, "Insult detection using a partitional CNN-LSTM model," *Computer Science and Information Technologies*, vol. 1, no. 2, pp. 84–92, 2020, doi: 10.11591/csit.v1i2.p84-92.

[11] E. Duman dan O. A. Erdem, "Anomaly Detection in Videos Using Optical Flow and Convolutional Autoencoder," *Institute of Electrical and Electronics Engineers Access*, vol. 7, pp. 183914–183923, 2019, doi: 10.1109/ACCESS.2019.2960654.

[12] B. Yang, J. Cao, R. Ni, dan L. Zou, "Anomaly Detection in Moving Crowds through Spatiotemporal Autoencoding and Additional Attention," *ADV Multimedia.*, vol. 2018, 2018, doi: 10.1155/2018/2087574.

[13] L. Zhang, G. Zhu, L. Mei, P. Shen, S. A. A. Shah, dan M. Bennamoun, "Attention in Convolutional LSTM for Gesture Recognition," *Advances in Neural Information Processing Systems*, vol. 31, 2018, [Online]. Tersedia: https://proceedings.neurips.cc/paper_files/paper/2018/file/287e03db1d99e0ec2edb90d079e142f3-Paper.pdf

[14] W. Ye, J. Cheng, F. Yang, dan Y. Xu, "Two-Stream Convolutional Network for Improving Activity Recognition Using Convolutional Long Short-Term Memory Networks," *Institute Of Electrical And Electronics Engineers Access*, vol. 7, pp. 67772–67780, 2019, doi: 10.1109/ACCESS.2019.2918808.

[15] S. Mobsite, N. Alaoui, M. Boulmalf, dan M. Ghogho, "Activity

Classification and Fall Detection using Monocular Depth and Motion Analysis," *Institute Of Electrical And Electronics Engineers Access*, no. January, pp. 1–1, 2023, doi: 10.1109/access.2023.3348413.

[16] Y. Li, Y. Tian, J. Tian, dan F. Zhou, "An Efficient Method for DPM Code Localization Based on Depthwise Separable Convolution," *Institute of Electrical and Electronics Engineers Access*, vol. 7, pp. 42014–42023, 2019, doi: 10.1109/ACCESS.2019.2905638.

[17] X. Wang, L. Yuan, H. Xu, dan X. Wen, "CSDS: End-to-End Aerial Scenes Classification with Depthwise Separable Convolution and an Attention Mechanism," *Institute of Electrical and Electronics Engineers Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 14, pp. 10484–10499, 2021, doi: 10.1109/JSTARS.2021.3117857.

[18] C. T. Liu, T. W. Lin, Y. H. Wu, Y. S. Lin, H. Lee, Y. Tsao, dan S. Y. Chien, "Computation-Performance Optimization of Convolutional Neural Networks with Redundant Filter Removal," *Institute of Electrical and Electronics Engineers Transactions on Circuits and Systems*, vol. 66, no. 5, pp. 1908–1921, 2019, doi: 10.1109/TCSI.2018.2885953.

[19] M. F. A. Abdullah, M. H. M. Hanafiah, S. Yogarayan, S. F. A. Razak, A. Azman, dan M. S. Sayeed, "Driver fatigue detection using Raspberry-Pi," *Indonesian Journal of Electrical Engineering and Computer Science*, vol. 32, no. 2, pp. 1142–1149, 2023, doi: 10.11591/ijeecs.v32.i2.pp1142-1149.

[20] I. Imanuddin, R. Maulana, dan M. Munawir, "Deteksi Mata Mengantuk Pada Pengemudi Mobil Menggunakan Metode Viola Jones," *Journal of Information Technology and Computer Science*, vol. 4, no. 2, p. 45, 2019, doi: 10.31328/jointecs.v4i2.1005.

[21] R. B. R. Putra dan H. Hendry, "Multivariate Time Series Forecasting pada Penjualan Barang Retail dengan Recurrent Neural Network," *The Journal of Innovation and Technology Polbeng Series on Informatics*, vol. 7, no. 1, p. 71, 2022, doi: 10.35314/isi.v7i1.2398.

[22] W. Wang, S. Hao, Y. Wei, S. Xiao, J. Feng, dan N. Sebe, "Temporal

Spiking Recurrent Neural Network for Action Recognition," *Institute of Electrical and Electronics Engineers Access*, vol. 7, pp. 117165–117175, 2019, doi: 10.1109/ACCESS.2019.2936604.

[23] C. G. Pachón-Suescún, J. O. Pinzón-Arenas, dan R. Jiménez-Moreno, "Abnormal gait detection by means of LSTM," *International Journal of Electrical and Computer Engineering*, vol. 10, no. 2, pp. 1495–1506, 2020, doi: 10.11591/ijece.v10i2.pp1495-1506.

[24] D. Yolanda, K. Gunadi, dan E. Setyati, "Pengenalan Alfabet Bahasa Isyarat Tangan Secara Real-Time dengan Menggunakan Metode Convolutional Neural Network dan Recurrent Neural Network," *Journal Informatika*, vol. 8, no. 1, pp. 203–208, 2020, [Online]. Tersedia: https://publication.petra.ac.id/index.php/teknik-informatika/article/view/9791

[25] Y. Widhiyasana, T. Semiawan, I. G. A. Mudzakir, dan M. R. Noor, "Penerapan Convolutional Long Short-Term Memory untuk Klasifikasi Teks Berita Bahasa Indonesia," *Jurnal Nasional Teknik Elektro dan Teknologi Informasi*, vol. 10, no. 4, pp. 354–361, 2021, doi: 10.22146/jnteti.v10i4.2438.

[26] M. D. Darojat, Y. A. Sari, dan R. C. Wihandika, "Convolutional Neural Network untuk Klasifikasi Citra Makanan Khas Indonesia," *Jurnal Pengembangan Teknologi Informasi dan Ilmu Komputer*, vol. 5, no. 11, pp. 4764–4769, 2021, [Online]. Tersedia: http://j-ptiik.ub.ac.id

[27] C. Kim, C. Kim, J. Bae, I. Baek, J. Jeong, Y. J. Lee, K. Park, S. H. Shim, dan S. B. Kim, "DESEM: Depthwise Separable Convolution-Based Multimodal Deep Learning for In-Game Action Anticipation," *Institute of Electrical and Electronics Engineers Access*, vol. 11, no. May, pp. 46504–46512, 2023, doi: 10.1109/ACCESS.2023.3271282.

[28] S. M. M. Nejad, D. Abbasi-Moghadam, A. Sharifi, N. Farmonov, K. Amankulova, dan M. Laszlz, "Multispectral Crop Yield Prediction Using 3D-Convolutional Neural Networks and Attention Convolutional LSTM

Approaches," *Institute of Electrical and Electronics Engineers Journal of Selected Topics in Applied Earth Observations and Remote Sensing.*, vol. 16, pp. 254–266, 2023, doi: 10.1109/JSTARS.2022.3223423.

[29] P. Sitompul, H. Okprana, A. Prasetio, dan G. Artikel, "Identifikasi Penyakit Tanaman Padi Melalui Citra Daun Menggunakan DenseNet 201 Identification of Rice Plant Diseases Through Leaf Image Using DenseNet 201 Article Info ABSTRAK," *Journal of Machine Learning and Artificial Intelligence*, vol. 1, no. 2, pp. 143–150, 2022, doi: 10.55123/jomlai.v1i2.889.

[30] V. P. Sharma, J. S. Yadav, dan V. Sharma, "Deep convolutional network based real time fatigue detection and drowsiness alertness system," *International Journal of Electrical and Computer Engineering*, vol. 12, no. 5, pp. 5493–5500, 2022, doi: 10.11591/ijece.v12i5.pp5493-5500.

[31] S. K. Mousavikia, E. Gholizadehazari, M. Mousazadeh, dan S. B. O. Yalcin, "Instruction Set Extension of a RiscV Based SoC for Driver Drowsiness Detection," *Institute of Electrical and Electronics Engineers Access*, vol. 10, pp. 58151–58162, 2022, doi: 10.1109/ACCESS.2022.3177743.

[32] E. K. Yılmaz dan M. A. Akcayol, "SUST-DDD: A Real-Drive Dataset for Driver Drowsiness Detection," *Proceeding 31st Conference of Open Innovations Association*, 2022.

[33] V. Vijaypriya dan M. Uma, "Facial Feature-Based Drowsiness Detection with Multi-Scale Convolutional Neural Network," *Institute of Electrical and Electronics Engineers Access*, vol. 11, no. June, pp. 63417–63429, 2023, doi: 10.1109/ACCESS.2023.3288008.

[34] L. Zhang, H. Saito, L. Yang, dan J. Wu, "Privacy-Preserving Federated Transfer Learning for Driver Drowsiness Detection," *Institute of Electrical and Electronics Engineers Access*, vol. 10, no. June, pp. 80565–80574, 2022, doi: 10.1109/ACCESS.2022.3192454.

# LAMPIRAN

## Lampiran 1. Code Program Implementasi

### 1.1. Persiapan

```
!pip3 install face_recognition
```

```
import os
import cv2
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
%matplotlib inline
from sklearn.model_selection import train_test_split
import joblib
from sklearn.model_selection import KFold
from sep_conv_rnn import SepConvLSTM2D
from sep_conv_rnn import ConvLSTM2D
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import ConvLSTM2D, MaxPooling3D, TimeDistributed,
Dropout, Flatten, Dense
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.utils import plot_model
from sklearn.metrics import confusion_matrix
import face_recognition
```

### 1.2. Menyiapkan Dataset

```
from google.colab import drive
drive.mount('/content/drive')
```

```
IMAGE_HEIGHT , IMAGE_WIDTH = 100, 100
SEQUENCE_LENGTH = 20
DATASET_DIR = "/content/drive/MyDrive/MBKM Riset 2023/Fajar/DriverDrowsiness"
all_classes_names = os.listdir(DATASET_DIR)
CLASSES_LIST = all_classes_names
```

### 1.3. Praproses dan Deteksi Wajah

```
def frames_extraction(video_path):
    frames_list = []
    video_reader = cv2.VideoCapture(video_path)
    video_frames_count = int(video_reader.get(cv2.CAP_PROP_FRAME_COUNT))
    skip_frames_window = max(int(video_frames_count/SEQUENCE_LENGTH), 1)
    for frame_counter in range(SEQUENCE_LENGTH):
        frame_counter * skip_frames_window)
```

```
            success, frame = video_reader.read()
            if not success:
                Break
            face_locations = face_recognition.face_locations(frame)
            if face_locations:
                top, right, bottom, left = face_locations[0]
                face_image = frame[top:bottom, left:right]
                resized_frame = cv2.resize(face_image, (IMAGE_HEIGHT,
                IMAGE_WIDTH))
                normalized_frame = resized_frame / 255
                frames_list.append(normalized_frame)
    video_reader.release()
    return frames_list
```

```
def create_dataset():
    features = []
    labels = []
    video_files_paths = []
    for class_index, class_name in enumerate(CLASSES_LIST):
        print(f'Extracting Data of Class: {class_name}')
        files_list = os.listdir(os.path.join(DATASET_DIR, class_name))
        for file_name in files_list:
            video_file_path = os.path.join(DATASET_DIR, class_name, file_name)
            frames = frames_extraction(video_file_path)
            if len(frames) == SEQUENCE_LENGTH:
                features.append(frames)
                labels.append(class_index)
                video_files_paths.append(video_file_path)
    features = np.asarray(features)
    labels = np.array(labels)
    return features, labels, video_files_paths
```

```
features, labels, video_files_paths = create_dataset()
```

### 1.4. Membagi dataset

```
one_hot_encoded_labels = to_categorical(labels)
indexfeatures_train, indexfeatures_test, labels_train, labels_test =
train_test_split(indexFeatures, one_hot_encoded_labels,test_size = 0.20, shuffle = True,
random_state = 27)
```

### 1.5. Membangun Arsitektur Model

```
def create_DSC_clstm_model():
    model = Sequential()
    model.add(SepConvLSTM2D(filters=32, kernel_size=(3,3), padding="same",
    return_sequences=True))
```

```
        model.add(MaxPooling3D(pool_size=(1, 2, 2), padding='same'))
        model.add(SepConvLSTM2D(filters=64, kernel_size=(3,3), padding="same",
        return_sequences=True))
        model.add(MaxPooling3D(pool_size=(1, 2, 2), padding='same'))
        model.add(Flatten())
        model.add(Dense(2, activation = "softmax"))
        model.build((None, 20, 100, 100, 3))
        model.summary()
return model
```

```
convlstm_model = create_convlstm_model()
```

### 1.6. Menentukan index data training dan validasi

```
k_folds = 5
kf = KFold(n_splits=k_folds)
```

```
y_train_fold, y_val_fold = labels_train[train_index], labels_train[val_index]
y_train_fold = np.argmax(y_train_fold, axis=1)
n_0=np.sum(y_train_fold[:] == 0)
n_1=np.sum(y_train_fold[:] == 1)
print(f"Train = Kelas 0 : {n_0} || Kelas 1 : {n_1}")
y_val_fold = np.argmax(y_val_fold, axis=1)
n_0=np.sum(y_val_fold[:] == 0)
n_1=np.sum(y_val_fold[:] == 1)
print(f"Val = Kelas 0 : {n_0} || Kelas 1 : {n_1}")
```

```
train_index, val_index=list(kf_loaded.split(features_train))[3]
X_train, X_val = features_train[train_index], features_train[val_index]
y_train, y_val = labels_train[train_index], labels_train[val_index]
```

### 1.7. Melakukan Pelatihan

```
DSC_clstm_model.compile(loss='categorical_crossentropy',
optimizer=Adam(learning_rate=0.00001), metrics=["accuracy"])
start_time = time.time()
DSC_clstm_model_training_history = DSC_clstm_model.fit(x=X_train, y=y_train,
epochs=100, batch_size=32,shuffle=True, validation_data=(X_val, y_val)
end_time = time.time()
execution_time = end_time - start_time
print("waktu training : ", execution_time, "detik")
tf.keras.saving.save_model(DSC_clstm_model,
f"/content/drive/MyDrive/skripsi/model/common2/saved_model_DSC_clstm_e100_lr0000
1.tf", overwrite=True, save_format="tf")
```

### 1.8. Evaluasi Model

```
def hitung_f1(labels_test, labels_pred):
    tp=0
    fp=0
    tn=0
    fn=0
    for index in range(len(labels_test1)):
        if labels_pred[index]==0 and labels_test[index]==0:
            tp+=1
        elif labels_pred[index]==0 and labels_test[index]==1:
            fp+=1
        elif labels_pred[index]==1 and labels_test[index]==1:
            tn+=1
        elif labels_pred[index]==1 and labels_test[index]==0:
            fn+=1
    precision=tp/(tp+fp)
    recall=tp/(tp+fn)
    f1=2*((precision*recall)/(precision+recall))
return precision, recall, f1
```

```
labels_pred=convlstm_model.predict(features_test)
labels_pred1 = np.argmax(labels_pred, axis=1)
labels_test1=np.argmax(labels_test, axis=1)
precision, recall, f1=hitung_f1(labels_test1, labels_pred1)
```

## Lampiran 2. Code Program SepConvLSTM2D

```
import numpy as np
import tensorflow as tf
from tensorflow.python.keras import activations
from tensorflow.python.keras import backend as K
from tensorflow.python.keras import constraints
from tensorflow.python.keras import initializers
from tensorflow.python.keras import regularizers
from tensorflow.python.keras.engine.base_layer import Layer
from tensorflow.python.keras.engine.input_spec import InputSpec
from tensorflow.python.keras.layers.recurrent import _standardize_args
from tensorflow.python.keras.layers.recurrent import DropoutRNNCellMixin
from tensorflow.python.keras.layers.recurrent import RNN
from tensorflow.python.keras.utils import conv_utils
from tensorflow.python.keras.utils import generic_utils
from tensorflow.python.keras.utils import tf_utils
from tensorflow.python.ops import array_ops
from tensorflow.python.util.tf_export import keras_export

class SepConvLSTM2DCell(DropoutRNNCellMixin, Layer):
  """Cell class for the SepConvLSTM2D layer.
  Arguments:
    filters: Integer, the dimensionality of the output space
      (i.e. the number of output filters in the convolution).
    kernel_size: An integer or tuple/list of n integers, specifying the
      dimensions of the convolution window.
    strides: An integer or tuple/list of n integers,
      specifying the strides of the convolution.
      Specifying any stride value != 1 is incompatible with specifying
      any `dilation_rate` value != 1.
    padding: One of `"valid"` or `"same"` (case-insensitive).
    data_format: A string,
      one of `channels_last` (default) or `channels_first`.
      It defaults to the `image_data_format` value found in your
      Keras config file at `~/.keras/keras.json`.
      If you never set it, then it will be "channels_last".
    dilation_rate: An integer or tuple/list of n integers, specifying
      the dilation rate to use for dilated convolution.
      Currently, specifying any `dilation_rate` value != 1 is
      incompatible with specifying any `strides` value != 1.
    depth_multiplier: The number of depthwise convolution output channels
      for each input channel. The total number of depthwise convolution output channels
      will be equal to filters_in * depth_multiplier
    activation: Activation function to use.
      If you don't specify anything, no activation is applied
      (ie. "linear" activation: `a(x) = x`).
```

```
  recurrent_activation: Activation function to use
    for the recurrent step.
  use_bias: Boolean, whether the layer uses a bias vector.
  kernel_initializer: Initializer for the `kernel` weights matrix,
    used for the linear transformation of the inputs.
  recurrent_initializer: Initializer for the `recurrent_kernel`
    weights matrix,
    used for the linear transformation of the recurrent state.
  bias_initializer: Initializer for the bias vector.
  unit_forget_bias: Boolean.
    If True, add 1 to the bias of the forget gate at initialization.
    Use in combination with `bias_initializer="zeros"`.
    This is recommended in [Jozefowicz et al.]
    (http://www.jmlr.org/proceedings/papers/v37/jozefowicz15.pdf)
  kernel_regularizer: Regularizer function applied to
    the `kernel` weights matrix.
  recurrent_regularizer: Regularizer function applied to
    the `recurrent_kernel` weights matrix.
  bias_regularizer: Regularizer function applied to the bias vector.
  kernel_constraint: Constraint function applied to
    the `kernel` weights matrix.
  recurrent_constraint: Constraint function applied to
    the `recurrent_kernel` weights matrix.
  bias_constraint: Constraint function applied to the bias vector.
  dropout: Float between 0 and 1.
    Fraction of the units to drop for
    the linear transformation of the inputs.
  recurrent_dropout: Float between 0 and 1.
    Fraction of the units to drop for
    the linear transformation of the recurrent state.
Call arguments:
  inputs: A 4D tensor.
  states:  List of state tensors corresponding to the previous timestep.
  training: Python boolean indicating whether the layer should behave in
    training mode or in inference mode. Only relevant when `dropout` or
    `recurrent_dropout` is used.
"""

def __init__(self,
             filters,
             kernel_size,
             strides=(1, 1),
             padding='valid',
             data_format=None,
             dilation_rate=(1, 1),
             depth_multiplier = 1,
             activation='tanh',
```

```
            recurrent_activation='hard_sigmoid',
            use_bias=True,
            kernel_initializer='glorot_uniform',
            recurrent_initializer='orthogonal',
            bias_initializer='zeros',
            unit_forget_bias=True,
            kernel_regularizer=None,
            recurrent_regularizer=None,
            bias_regularizer=None,
            kernel_constraint=None,
            recurrent_constraint=None,
            bias_constraint=None,
            dropout=0.,
            recurrent_dropout=0.,
            **kwargs):
    super(SepConvLSTM2DCell, self).__init__(**kwargs)
    self.filters = filters
    self.kernel_size = conv_utils.normalize_tuple(kernel_size, 2, 'kernel_size')
    self.strides = conv_utils.normalize_tuple(strides, 2, 'strides')
    self.padding = conv_utils.normalize_padding(padding)
    self.data_format = conv_utils.normalize_data_format(data_format)
    self.dilation_rate = conv_utils.normalize_tuple(dilation_rate, 2,
                                                    'dilation_rate')
    self.depth_multiplier = depth_multiplier
    self.activation = activations.get(activation)
    self.recurrent_activation = activations.get(recurrent_activation)
    self.use_bias = use_bias

    self.kernel_initializer = initializers.get(kernel_initializer)
    self.recurrent_initializer = initializers.get(recurrent_initializer)
    self.bias_initializer = initializers.get(bias_initializer)
    self.unit_forget_bias = unit_forget_bias

    self.kernel_regularizer = regularizers.get(kernel_regularizer)
    self.recurrent_regularizer = regularizers.get(recurrent_regularizer)
    self.bias_regularizer = regularizers.get(bias_regularizer)

    self.kernel_constraint = constraints.get(kernel_constraint)
    self.recurrent_constraint = constraints.get(recurrent_constraint)
    self.bias_constraint = constraints.get(bias_constraint)

    self.dropout = min(1., max(0., dropout))
    self.recurrent_dropout = min(1., max(0., recurrent_dropout))
    self.state_size = (self.filters, self.filters)

def build(self, input_shape):
```

```
if self.data_format == 'channels_first':
  channel_axis = 1
else:
  channel_axis = -1
if input_shape[channel_axis] is None:
  raise ValueError('The channel dimension of the inputs '
           'should be defined. Found `None`.')
input_dim = input_shape[channel_axis]
depth_kernel_shape = self.kernel_size + ( input_dim, self.depth_multiplier * 4)
point_kernel_shape = (1,1) + ( input_dim * self.depth_multiplier, self.filters * 4)

self.depth_kernel_shape = depth_kernel_shape
self.point_kernel_shape = point_kernel_shape

recurrent_depth_kernel_shape = self.kernel_size + ( self.filters , self.depth_multiplier * 4 )
recurrent_point_kernel_shape = (1,1) + ( self.filters * self.depth_multiplier , self.filters * 4 )

self.depth_kernel_shape = depth_kernel_shape
self.point_kernel_shape = point_kernel_shape


self.depth_kernel = self.add_weight(shape=depth_kernel_shape,
                    initializer=self.kernel_initializer,
                    name='depth_kernel',
                    regularizer=self.kernel_regularizer,
                    constraint=self.kernel_constraint)

self.point_kernel = self.add_weight(shape=point_kernel_shape,
                    initializer=self.kernel_initializer,
                    name='point_kernel',
                    regularizer=self.kernel_regularizer,
                    constraint=self.kernel_constraint)


self.recurrent_depth_kernel = self.add_weight(
                    shape=recurrent_depth_kernel_shape,
                    initializer=self.recurrent_initializer,
                    name='recurrent_depth_kernel',
                    regularizer=self.recurrent_regularizer,
                    constraint=self.recurrent_constraint)

self.recurrent_point_kernel = self.add_weight(
                    shape=recurrent_point_kernel_shape,
                    initializer=self.recurrent_initializer,
                    name='recurrent_point_kernel',
                    regularizer=self.recurrent_regularizer,
                    constraint=self.recurrent_constraint)
```

```
  if self.use_bias:
    if self.unit_forget_bias:

      def bias_initializer(_, *args, **kwargs):
        return K.concatenate([
            self.bias_initializer((self.filters,), *args, **kwargs),
            initializers.Ones()((self.filters,), *args, **kwargs),
            self.bias_initializer((self.filters * 2,), *args, **kwargs),
        ])
    else:
      bias_initializer = self.bias_initializer
    self.bias = self.add_weight(
        shape=(self.filters * 4,),
        name='bias',
        initializer=bias_initializer,
        regularizer=self.bias_regularizer,
        constraint=self.bias_constraint)
  else:
    self.bias = None
  self.built = True

def call(self, inputs, states, training=None):
  h_tm1 = states[0]  # previous memory state
  c_tm1 = states[1]  # previous carry state

  # dropout matrices for input units
  dp_mask = self.get_dropout_mask_for_cell(inputs, training, count=4)
  # dropout matrices for recurrent units
  rec_dp_mask = self.get_recurrent_dropout_mask_for_cell(
      h_tm1, training, count=4)

  if 0 < self.dropout < 1.:
    inputs_i = inputs * dp_mask[0]
    inputs_f = inputs * dp_mask[1]
    inputs_c = inputs * dp_mask[2]
    inputs_o = inputs * dp_mask[3]
  else:
    inputs_i = inputs
    inputs_f = inputs
    inputs_c = inputs
    inputs_o = inputs

  if 0 < self.recurrent_dropout < 1.:
    h_tm1_i = h_tm1 * rec_dp_mask[0]
    h_tm1_f = h_tm1 * rec_dp_mask[1]
```

```
    h_tm1_c = h_tm1 * rec_dp_mask[2]
    h_tm1_o = h_tm1 * rec_dp_mask[3]
  else:
    h_tm1_i = h_tm1
    h_tm1_f = h_tm1
    h_tm1_c = h_tm1
    h_tm1_o = h_tm1

  (depth_kernel_i, depth_kernel_f,
   depth_kernel_c, depth_kernel_o) = array_ops.split(self.depth_kernel, 4, axis=3)
  (recurrent_depth_kernel_i,
   recurrent_depth_kernel_f,
   recurrent_depth_kernel_c,
   recurrent_depth_kernel_o) = array_ops.split(self.recurrent_depth_kernel, 4, axis=3)

  (point_kernel_i, point_kernel_f,
   point_kernel_c, point_kernel_o) = array_ops.split(self.point_kernel, 4, axis=3)
  (recurrent_point_kernel_i,
   recurrent_point_kernel_f,
   recurrent_point_kernel_c,
   recurrent_point_kernel_o) = array_ops.split(self.recurrent_point_kernel, 4, axis=3)

  if self.use_bias:
    bias_i, bias_f, bias_c, bias_o = array_ops.split(self.bias, 4)
  else:
    bias_i, bias_f, bias_c, bias_o = None, None, None, None

  x_i = self.input_conv(inputs_i, depth_kernel_i, point_kernel_i, bias_i, padding=self.padding)
  x_f = self.input_conv(inputs_f, depth_kernel_f, point_kernel_f, bias_f, padding=self.padding)
  x_c = self.input_conv(inputs_c, depth_kernel_c, point_kernel_c, bias_c, padding=self.padding)
  x_o = self.input_conv(inputs_o, depth_kernel_o, point_kernel_o, bias_o, padding=self.padding)
  h_i = self.recurrent_conv(h_tm1_i, recurrent_depth_kernel_i, recurrent_point_kernel_i)
  h_f = self.recurrent_conv(h_tm1_f, recurrent_depth_kernel_f, recurrent_point_kernel_f)
  h_c = self.recurrent_conv(h_tm1_c, recurrent_depth_kernel_c, recurrent_point_kernel_c)
  h_o = self.recurrent_conv(h_tm1_o, recurrent_depth_kernel_o, recurrent_point_kernel_o)

  i = self.recurrent_activation(x_i + h_i)
  f = self.recurrent_activation(x_f + h_f)
  c = f * c_tm1 + i * self.activation(x_c + h_c)
  o = self.recurrent_activation(x_o + h_o)
  h = o * self.activation(c)
  return h, [h, c]

def input_conv(self, x, dw, pw, b=None, padding='valid'):
  conv_out = K.separable_conv2d(x, dw, pw, strides=self.strides,
                                padding=padding,
                                data_format=self.data_format,
```

```
                    dilation_rate=self.dilation_rate)
    if b is not None:
      conv_out = K.bias_add(conv_out, b,
                            data_format=self.data_format)
    return conv_out

  def recurrent_conv(self, x, dw, pw):
    conv_out = K.separable_conv2d(x, dw, pw, strides=(1, 1),
                    padding='same',
                    data_format=self.data_format)
    return conv_out

  def get_config(self):
    config = {'filters': self.filters,
              'kernel_size': self.kernel_size,
              'strides': self.strides,
              'padding': self.padding,
              'data_format': self.data_format,
              'dilation_rate': self.dilation_rate,
              'depth_multiplier':self.depth_multiplier,
              'activation': activations.serialize(self.activation),
              'recurrent_activation': activations.serialize(
                self.recurrent_activation),
              'use_bias': self.use_bias,
              'kernel_initializer': initializers.serialize(
                self.kernel_initializer),
              'recurrent_initializer': initializers.serialize(
                self.recurrent_initializer),
              'bias_initializer': initializers.serialize(self.bias_initializer),
              'unit_forget_bias': self.unit_forget_bias,
              'kernel_regularizer': regularizers.serialize(
                self.kernel_regularizer),
              'recurrent_regularizer': regularizers.serialize(
                self.recurrent_regularizer),
              'bias_regularizer': regularizers.serialize(self.bias_regularizer),
              'kernel_constraint': constraints.serialize(
                self.kernel_constraint),
              'recurrent_constraint': constraints.serialize(
                self.recurrent_constraint),
              'bias_constraint': constraints.serialize(self.bias_constraint),
              'dropout': self.dropout,
              'recurrent_dropout': self.recurrent_dropout}
    base_config = super(SepConvLSTM2DCell, self).get_config()
    return dict(list(base_config.items()) + list(config.items()))


@keras_export('keras.layers.SepConvLSTM2D')
```

```
class SepConvLSTM2D(SepConvRNN2D):
 """Seperable Convolutional LSTM.
 It is similar to an LSTM layer, but the input transformations
 and recurrent transformations are both depthwise seperable convolutional.
 Arguments:
  filters: Integer, the dimensionality of the output space
   (i.e. the number of output filters in the convolution).
  kernel_size: An integer or tuple/list of n integers, specifying the
   dimensions of the convolution window.
  strides: An integer or tuple/list of n integers,
   specifying the strides of the convolution.
   Specifying any stride value != 1 is incompatible with specifying
   any `dilation_rate` value != 1.
  padding: One of `"valid"` or `"same"` (case-insensitive).
  data_format: A string,
   one of `channels_last` (default) or `channels_first`.
   The ordering of the dimensions in the inputs.
   `channels_last` corresponds to inputs with shape
   `(batch, time, ..., channels)`
   while `channels_first` corresponds to
   inputs with shape `(batch, time, channels, ...)`.
   It defaults to the `image_data_format` value found in your
   Keras config file at `~/.keras/keras.json`.
   If you never set it, then it will be "channels_last".
  dilation_rate: An integer or tuple/list of n integers, specifying
   the dilation rate to use for dilated convolution.
   Currently, specifying any `dilation_rate` value != 1 is
   incompatible with specifying any `strides` value != 1.
  depth_multiplier: The number of depthwise convolution output channels
   for each input channel. The total number of depthwise convolution output channels
   will be equal to filters_in * depth_multiplier
  activation: Activation function to use.
   By default hyperbolic tangent activation function is applied
   (`tanh(x)`).
  recurrent_activation: Activation function to use
   for the recurrent step.
  use_bias: Boolean, whether the layer uses a bias vector.
  kernel_initializer: Initializer for the `kernel` weights matrix,
   used for the linear transformation of the inputs.
  recurrent_initializer: Initializer for the `recurrent_kernel`
   weights matrix,
   used for the linear transformation of the recurrent state.
  bias_initializer: Initializer for the bias vector.
  unit_forget_bias: Boolean.
   If True, add 1 to the bias of the forget gate at initialization.
   Use in combination with `bias_initializer="zeros"`.
   This is recommended in [Jozefowicz et al.]
```

```
    (http://www.jmlr.org/proceedings/papers/v37/jozefowicz15.pdf)
  kernel_regularizer: Regularizer function applied to
    the `kernel` weights matrix.
  recurrent_regularizer: Regularizer function applied to
    the `recurrent_kernel` weights matrix.
  bias_regularizer: Regularizer function applied to the bias vector.
  activity_regularizer: Regularizer function applied to.
  kernel_constraint: Constraint function applied to
    the `kernel` weights matrix.
  recurrent_constraint: Constraint function applied to
    the `recurrent_kernel` weights matrix.
  bias_constraint: Constraint function applied to the bias vector.
  return_sequences: Boolean. Whether to return the last output
    in the output sequence, or the full sequence.
  go_backwards: Boolean (default False).
    If True, process the input sequence backwards.
  stateful: Boolean (default False). If True, the last state
    for each sample at index i in a batch will be used as initial
    state for the sample of index i in the following batch.
  dropout: Float between 0 and 1.
    Fraction of the units to drop for
    the linear transformation of the inputs.
  recurrent_dropout: Float between 0 and 1.
    Fraction of the units to drop for
    the linear transformation of the recurrent state.
Call arguments:
  inputs: A 5D tensor.
  mask: Binary tensor of shape `(samples, timesteps)` indicating whether
    a given timestep should be masked.
  training: Python boolean indicating whether the layer should behave in
    training mode or in inference mode. This argument is passed to the cell
    when calling it. This is only relevant if `dropout` or `recurrent_dropout`
    are set.
  initial_state: List of initial state tensors to be passed to the first
    call of the cell.
Input shape:
  - If data_format='channels_first'
      5D tensor with shape:
      `(samples, time, channels, rows, cols)`
  - If data_format='channels_last'
      5D tensor with shape:
      `(samples, time, rows, cols, channels)`
Output shape:
  - If `return_sequences`
     - If data_format='channels_first'
        5D tensor with shape:
        `(samples, time, filters, output_row, output_col)`
```

```
    - If data_format='channels_last'
      5D tensor with shape:
      `(samples, time, output_row, output_col, filters)`
  - Else
    - If data_format ='channels_first'
      4D tensor with shape:
      `(samples, filters, output_row, output_col)`
    - If data_format='channels_last'
      4D tensor with shape:
      `(samples, output_row, output_col, filters)`
    where `o_row` and `o_col` depend on the shape of the filter and
    the padding
Raises:
  ValueError: in case of invalid constructor arguments.
References:
  - [Convolutional LSTM Network: A Machine Learning Approach for
  Precipitation Nowcasting](http://arxiv.org/abs/1506.04214v1)
  The current implementation does not include the feedback loop on the
  cells output.
"""

def __init__(self,
             filters,
             kernel_size,
             strides=(1, 1),
             padding='valid',
             data_format=None,
             dilation_rate=(1, 1),
             depth_multiplier = 1,
             activation='tanh',
             recurrent_activation='hard_sigmoid',
             use_bias=True,
             kernel_initializer='glorot_uniform',
             recurrent_initializer='orthogonal',
             bias_initializer='zeros',
             unit_forget_bias=True,
             kernel_regularizer=None,
             recurrent_regularizer=None,
             bias_regularizer=None,
             activity_regularizer=None,
             kernel_constraint=None,
             recurrent_constraint=None,
             bias_constraint=None,
             return_sequences=False,
             go_backwards=False,
             stateful=False,
             dropout=0.,
```

```
            recurrent_dropout=0.,
            **kwargs):
  cell = SepConvLSTM2DCell(filters=filters,
                           kernel_size=kernel_size,
                           strides=strides,
                           padding=padding,
                           data_format=data_format,
                           dilation_rate=dilation_rate,
                           depth_multiplier=depth_multiplier,
                           activation=activation,
                           recurrent_activation=recurrent_activation,
                           use_bias=use_bias,
                           kernel_initializer=kernel_initializer,
                           recurrent_initializer=recurrent_initializer,
                           bias_initializer=bias_initializer,
                           unit_forget_bias=unit_forget_bias,
                           kernel_regularizer=kernel_regularizer,
                           recurrent_regularizer=recurrent_regularizer,
                           bias_regularizer=bias_regularizer,
                           kernel_constraint=kernel_constraint,
                           recurrent_constraint=recurrent_constraint,
                           bias_constraint=bias_constraint,
                           dropout=dropout,
                           recurrent_dropout=recurrent_dropout,
                           dtype=kwargs.get('dtype'))
  super(SepConvLSTM2D, self).__init__(cell,
                                      return_sequences=return_sequences,
                                      go_backwards=go_backwards,
                                      stateful=stateful,
                                      **kwargs)
  self.activity_regularizer = regularizers.get(activity_regularizer)

def call(self, inputs, mask=None, training=None, initial_state=None):
  self._maybe_reset_cell_dropout_mask(self.cell)
  return super(SepConvLSTM2D, self).call(inputs,
                                         mask=mask,
                                         training=training,
                                         initial_state=initial_state)

@property
def filters(self):
  return self.cell.filters

@property
def kernel_size(self):
  return self.cell.kernel_size
```

```
@property
def strides(self):
  return self.cell.strides

@property
def padding(self):
  return self.cell.padding

@property
def data_format(self):
  return self.cell.data_format

@property
def dilation_rate(self):
  return self.cell.dilation_rate

@property
def depth_multiplier(self):
  return self.cell.depth_multiplier

@property
def activation(self):
  return self.cell.activation

@property
def recurrent_activation(self):
  return self.cell.recurrent_activation

@property
def use_bias(self):
  return self.cell.use_bias

@property
def kernel_initializer(self):
  return self.cell.kernel_initializer

@property
def recurrent_initializer(self):
  return self.cell.recurrent_initializer

@property
def bias_initializer(self):
  return self.cell.bias_initializer

@property
def unit_forget_bias(self):
  return self.cell.unit_forget_bias
```

```
@property
def kernel_regularizer(self):
  return self.cell.kernel_regularizer

@property
def recurrent_regularizer(self):
  return self.cell.recurrent_regularizer

@property
def bias_regularizer(self):
  return self.cell.bias_regularizer

@property
def kernel_constraint(self):
  return self.cell.kernel_constraint

@property
def recurrent_constraint(self):
  return self.cell.recurrent_constraint

@property
def bias_constraint(self):
  return self.cell.bias_constraint

@property
def dropout(self):
  return self.cell.dropout

@property
def recurrent_dropout(self):
  return self.cell.recurrent_dropout

def get_config(self):
  config = {'filters': self.filters,
            'kernel_size': self.kernel_size,
            'strides': self.strides,
            'padding': self.padding,
            'data_format': self.data_format,
            'dilation_rate': self.dilation_rate,
            'depth_multiplier':self.depth_multiplier,
            'activation': activations.serialize(self.activation),
            'recurrent_activation': activations.serialize(
              self.recurrent_activation),
            'use_bias': self.use_bias,
            'kernel_initializer': initializers.serialize(
              self.kernel_initializer),
```

```
            'recurrent_initializer': initializers.serialize(
                self.recurrent_initializer),
            'bias_initializer': initializers.serialize(self.bias_initializer),
            'unit_forget_bias': self.unit_forget_bias,
            'kernel_regularizer': regularizers.serialize(
                self.kernel_regularizer),
            'recurrent_regularizer': regularizers.serialize(
                self.recurrent_regularizer),
            'bias_regularizer': regularizers.serialize(self.bias_regularizer),
            'activity_regularizer': regularizers.serialize(
                self.activity_regularizer),
            'kernel_constraint': constraints.serialize(
                self.kernel_constraint),
            'recurrent_constraint': constraints.serialize(
                self.recurrent_constraint),
            'bias_constraint': constraints.serialize(self.bias_constraint),
            'dropout': self.dropout,
            'recurrent_dropout': self.recurrent_dropout}
        base_config = super(SepConvLSTM2D, self).get_config()
        del base_config['cell']
        return dict(list(base_config.items()) + list(config.items()))

    @classmethod
    def from_config(cls, config):
        return cls(**config)
```