TUGAS AKHIR – EE184801

DETEKSI LAJUR UNTUK MOBIL OTONOM PADA KONDISI LINGKUNGAN PENCAHAYAAN RENDAH DAN TERDISTORSI BERBASIS HYBRID CNN-RNN

MUHAMMAD CENDEKIA AIRLANGGA

NRP 07111840000077

Dosen Pembimbing

Dr.Ir. Ari Santoso, DEA.

NIP 196602181991021001

Yusuf Bilfaqih, S.T., M.T.

NIP 197203251999031001

Program Studi Sarjana Teknik Elektro

Departemen Teknik Elektro

Fakultas Teknologi Elektro dan Informatika Cerdas

Institut Teknologi Sepuluh Nopember

Surabaya

2022

TUGAS AKHIR – EE184801

DETEKSI LAJUR UNTUK MOBIL OTONOM PADA KONDISI LINGKUNGAN PENCAHAYAAN RENDAH DAN TERDISTORSI BERBASIS HYBRID CNN-RNN

MUHAMMAD CENDEKIA AIRLANGGA

NRP 07111840000077

Dosen Pembimbing

Dr. Ir. Ari Santoso, DEA.

NIP 196602181991021001

Yusuf Bilfaqih, S.T., M.T.

NIP 197203251999031001

Program Studi Sarjana Teknik Elektro

Departemen Teknik Elektro

Fakultas Teknologi Elektro dan Informatika Cerdas

Institut Teknologi Sepuluh Nopember

Surabaya

2022

FINAL PROJECT – EE184801

LANE DETECTION FOR AUTONOMOUS CAR IN LOW-LIGHT AND DISTORTED ENVIRONMENT BASED ON HYBRID CNN-RNN

MUHAMMAD CENDEKIA AIRLANGGA

NRP 07111840000077

Advisor

Dr. Ir. Ari Santoso, DEA.

NIP 196602181991021001

Yusuf Bilfaqih, S.T., M.T.

NIP 197203251999031001

Electrical Engineering Undergraduate Program

Department of Electrical Engineering

Faculty of Intelligent Electrical and Informatics Technology

Institut Teknologi Sepuluh Nopember

Surabaya

2022

LEMBAR PENGESAHAN

DETEKSI LAJUR UNTUK MOBIL OTONOM PADA KONDISI LINGKUNGAN PENCAHAYAAN RENDAH DAN TERDISTORSI BERBASIS HYBRID CNN-RNN

TUGAS AKHIR

Diajukan untuk memenuhi salah satu syarat
memperoleh gelar Sarjana pada
Program Studi S-1 Teknik Elektro
Departemen Teknik Elektro
Fakultas Teknologi Elektronika dan Informatika Cerdas
Institut Teknologi Sepuluh Nopember

Oleh: **MUHAMMAD CENDEKIA AIRLANGGA**
NRP. 07111840000077

Disetujui oleh Tim Penguji Tugas Akhir:

1. Dr. Ir. Ari Santoso, DEA. — Pembimbing
2. Yusuf Bilfaqih, S.T., M.T. — Ko-pembimbing
3. Prof. Ir. Abdullah Alkaff, M.Sc., Ph.D — Penguji
4. Dr. Trihastuti Agustinah, ST., MT. — Penguji
5. Mochammad Sahal, ST., M.Sc. — Penguji

SURABAYA
Juli, 2022

Halaman ini sengaja dikosongkan

APPROVAL SHEET

LANE DETECTION FOR AUTONOMOUS CAR IN LOW-LIGHT AND DISTORTED ENVIRONMENT BASED ON HYBRID CNN-RNN

FINAL PROJECT

Submitted to fulfill one of the requirements
for obtaining a bachelor degree at
Undergraduate Study Program of Electrical Engineering
Department of Electrical Engineering
Faculty of Intelligent Electrical and Informatics Technology
Institut Teknologi Sepuluh Nopember

By: **MUHAMMAD CENDEKIA AIRLANGGA**
NRP. 07111840000077
Approved by Final Project Examiner Team:

1. Dr. Ir. Ari Santoso, DEA. — Advisor
2. Yusuf Bilfaqih, S.T., M.T. — Co-Advisor
3. Prof. Ir. Abdullah Alkaff, M.Sc., Ph.D — Examiner
4. Dr. Trihastuti Agustinah, ST., MT. — Examiner
5. Mochammad Sahal, ST., M.Sc. — Examiner

SURABAYA
July, 2022

Halaman ini sengaja dikosongkan

PERNYATAAN ORISINALITAS

Yang bertanda tangan di bawah ini:

Nama mahasiswa / NRP	:	Muhammad Cendekia Airlangga / 07111840000077
Program studi	:	Teknik Elektro
Dosen Pembimbing / NIP	:	Dr. Ir. Ari Santoso, DEA. / 196602181991021001

dengan ini menyatakan bahwa Tugas Akhir dengan judul "Deteksi Lajur untuk Mobil Otonom Pada Kondisi Lingkungan Pencahayaan Rendah dan Terdistorsi Berbasis Hybrid CNN-RNN" adalah hasil karya sendiri, bersifat orisinal, dan ditulis dengan mengikuti kaidah penulisan ilmiah.

Bilamana di kemudian hari ditemukan ketidaksesuaian dengan pernyataan ini, maka saya bersedia menerima sanksi sesuai dengan ketentuan yang berlaku di Institut Teknologi Sepuluh Nopember.

Surabaya, 11 Juni 2022

Mengetahui
Dosen Pembimbing

Dr. Ir. Ari Santoso, DEA.
NIP. 196602181991021001

Mahasiswa

Muhammad Cendekia Airlangga
NRP. 07111840000077

Halaman ini sengaja dikosongkan

STATEMENT OF ORIGINALITY

The undersigned below:

Name of student / NRP	:	Muhammad Cendekia Airlangga / 07111840000077
Department	:	Electrical Engineering
Advisor / NIP	:	Dr. Ir. Ari Santoso, DEA. / 196602181991021001

Hereby declare that the Final Project with the title of "Lane Detection for Autonomous Car In Low-Light And Distorted Environment Based On Hybrid CNN-RNN" is the result of my own work, is original, and is written by following the rules of scientific writing.

If in the future there is a discrepancy with this statement, then I am willing to accept sanctions in accordance with the provisions that apply at Institut Teknologi Sepuluh Nopember.

Surabaya, June 11th 2022

Acknowledged
Advisor

Dr. Ir. Ari Santoso, DEA.
NIP. 196602181991021001

Student

Muhammad Cendekia Airlangga
NRP. 07111840000077

Halaman ini sengaja dikosongkan

ABSTRAK

DETEKSI LAJUR UNTUK MOBIL OTONOM PADA KONDISI LINGKUNGAN PENCAHAYAAN RENDAH DAN TERDISTORSI BERBASIS HYBRID CNN-RNN

Nama Mahasiswa / NRP : Muhammad Cendekia Airlangga / 07111840000077
Departemen : Teknik Elektro FTEIC-ITS
Dosen Pembimbing : Dr. Ir. Ari Santoso, DEA.

Abstrak

Deteksi lajur termasuk dalam *Driving scene understanding*: kemampuan mobil otonom dalam mempersepsikan lingkungan di sekitarnya tanpa bantuan manusia. Untuk melakukan deteksi ini, algoritma seperti *hough transform* dan *convolutional neural network* (CNN) telah dikembangkan. Algoritma tersebut mengalami penurunan performa yang signifikan dalam situasi tertentu seperti cuaca hujan, malam hari. Penurunan performa dari *hough transform* ini terjadi karena beberapa faktor: *fixed region of interest* (ROI) yang menyebabkan algoritma tidak bisa mendeteksi di luar ROI, hasil deteksi yang tidak adaptif, dan hasil deteksi yang terbatas hanya untuk dua garis lajur. Sementara itu, CNN memiliki kelemahan yaitu deteksi hanya dilakukan pada *current frame* saja. Pada penelitian ini, metode *hybrid* CNN-RNN digunakan untuk menyelesaikan permasalahan deteksi lajur. Kombinasi CNN-RNN ini bertujuan untuk memanfaatkan kemampuan ekstraksi fitur dari CNN dan pengolahan informasi temporal dari RNN. Dua arsitektur digunakan yaitu SegNet-ConvLSTM dan UNet-ConvLSTM. Pengujian performa dalam bentuk F1-*score* kedua arsitektur ini dilakukan untuk beberapa kondisi cuaca dan jalan. Secara rata-rata, UNet-ConvLSTM memiliki performa yang lebih baik. Namun, pada saat kondisi hujan, SegNet-ConvLSTM memiliki performa yang lebih baik. *Hough transform* menghasilkan nilai F1-*score* yang paling rendah. Penggunaan ConvLSTM memiliki pengaruh untuk mencegah penurunan performa yang signifikan pada saat Hujan dan Malam Hari. Pengujian terkait *running time* juga dilakukan pada penelitian ini. *Running time* dari SegNet-ConvLSTM dan UNet-ConvLSTM secara berurutan adalah 0.0327 detik dan 0.0281 detik. Nilai ini menunjukkan bahwa algoritma dapat bekerja secara *realtime* dan jauh lebih cepat dibandingkan dengan *hough transform* yang memiliki *running time* sebesar 0.0401 detik.

Kata kunci: *Hybrid CNN-RNN, Mobil Otonom, Deteksi Lajur, Deep Learning*

Halaman ini sengaja dikosongkan

ABSTRACT

LANE DETECTION FOR AUTONOMOUS CAR IN LOW-LIGHT AND DISTORTED ENVIRONMENT BASED ON HYBRID CNN-RNN

Student Name / NRP	**:**	**Muhammad Cendekia Airlangga / 07111840000077**
Department	**:**	**Teknik Elektro FTEIC-ITS**
Advisor	**:**	**Dr. Ir. Ari Santoso, DEA.**

Abstract

Lane detection is one of important parts from Driving scene understanding: the ability of an autonomous car to perceive its surroundings without human assistance. Several algorithms such as the Hough transform and convolutional neural network (CNN) have been developed. The algorithm suffers from a significant decrease in performance in certain situations such as rainy weather, night. This significant performance drop in Hough transformation occurs due to several factors: fixed region of interest (ROI) which causes the algorithm to be unable to detect outside the ROI, non-adaptive detection results, and detection results that are limited to only two lanes. Meanwhile, CNN's detection is only done on the current frame. In this study, the CNN-RNN hybrid method was used to perform a lane detection. The combination of CNN-RNN aims to take advantage of the feature extraction capabilities of CNN and temporal information processing of RNN. Two architectures used are SegNet-ConvLSTM and UNet-ConvLSTM. Performance testing in the form of F1-scores of these two architectures was carried out for several weather and road conditions. On average, UNet-ConvLSTM performs better. However, during rainy conditions, SegNet-ConvLSTM has better performance. Hough transformation produces the lowest F1-score value. The use of ConvLSTM has the effect of preventing a significant decrease in performance during Rain and Night. Tests related to running time were also carried out in this study. The running times of SegNet-ConvLSTM and UNet-ConvLSTM are 0.0327 seconds and 0.0281 seconds, respectively. This value indicates that the algorithm can work in real time and is much faster than the hough transform which has a running time of 0.0401 seconds.

Keywords: ***Hybrid CNN-RNN, Autonomous Car, Lane Detection, Deep Learning***

Halaman ini sengaja dikosongkan

KATA PENGANTAR

Segala Puji syukur penulis panjatkan kepada Allah S.W.T yang telah memberikan rahmat dan karunianya. Shalawat beriring salam tak lupa juga senantiasa penulis ucapkan kepada junjungan kita Nabi Muhammad S.A.W, sehingga penulis dapat membuat dan menyelesaikan tugas akhir ini dengan baik dan tepat waktu.

Kegiatan tugas akhir ini merupakan bagian dari penyelesaian masa studi di S-1 Departemen Teknik Elektro Institut Teknologi Sepuluh Nopember. Tugas akhir yang penulis lakukan mengangkat topik Deteksi Lajur untuk Mobil Otonom dengan menggunakan metode *hybrid* CNN-RNN. Laporan tugas akhir ini disusun untuk melengkapi hasil capaian dari tugas akhir yang telah dilaksanakan.

Dalam pembuatan dan penyelesaian tugas akhir ini, penulis telah dibantu dan didukung oleh banyak pihak. Oleh karena itu, penulis menyampaikan terimakasih kepada:

1. Orang tua serta keluarga penulis yang telah memberi dukungan dan doa kepada penulis.

2. Bapak Dr. Ir. Ari Santoso, DEA dan Bapak Yusuf Bilfaqih, S.T., M.T. sebagai dosen pembimbing tugas akhir penulis yang telah memberi arahan dan bimbingan kepada penulis selama proses pengerjaan tugas akhir ini.

3. Tenaga pendidik dan dosen Departemen Teknik Elektro ITS, khususnya bidang studi Teknik Sistem Pengaturan.

4. Teman-teman penulis yang telah memberi dukungan teknis maupun non teknis selama proses pengerjaan.

Penulis menyadari bahwa masih terdapat kekurangan dari laporan tugas akhir ini, mengingat masih kurangnya pengetahuan dan pengalaman penulis. Penulis berharap tugas akhir ini dapat bermanfaat bagi pembaca. Saran dan kritik juga penulis harapkan untuk pengembangan tugas akhir ini.

Surabaya, 11 Juni 2022

Muhammad Cendekia Airlangga

Halaman ini sengaja dikosongkan

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

Halaman ini sengaja dikosongkan

BAB 1 PENDAHULUAN

1.1 Latar Belakang

Dunia otomotif mengalami perkembangan yang pesat saat ini. Salah satu perkembangan yang terjadi di dunia otomotif adalah adanya mobil otonom. Mobil otonom merupakan salah satu bidang pengembangan teknologi yang banyak diminati baik dari kalangan industri maupun akademisi. Mobil otonom dapat didefinisikan sebagai suatu kendaraan yang memiliki kemampuan untuk mempersepsikan lingkungan sekitarnya dan mampu untuk melakukan navigasi tanpa campur tangan manusia (Jo et al., 2014). Kemampuan untuk mempersepsikan lingkungan sekitar dengan menggunakan sensor-sensor seperti kamera, radar, dan lidar ini disebut dengan *driving scene understanding*.

Salah satu aspek penting dalam *driving scene understanding* adalah deteksi lajur pada jalan. Suatu lajur pada jalan dapat didefinisikan sebagai suatu struktur garis yang kontinu, baik garis yang solid maupun garis putus-putus yang berfungsi untuk mengarahkan arus kendaraan (Zou et al., 2019). Deteksi lajur yang baik memainkan peranan penting karena pengetahuan tentang posisi lajur dapat membantu kendaraan untuk mengetahui ke arah mana kendaraan harus berjalan dan mencegah kendaraan tersebut untuk mengambil posisi yang salah. Keberhasilan proses deteksi lajur dari suatu mobil otonom dipengaruhi oleh beberapa faktor, seperti tingkat iluminasi jalan dan kondisi cuaca (Tang et al., 2021).

Permasalahan yang terkait dengan deteksi lajur termasuk dalam bidang kajian *computer vision.* Salah satu algoritma dalam bidang *computer vision* yang telah digunakan dalam permasalahan deteksi lajur adalah *hough transform* (Widianto, 2020). Algoritma ini mampu menghasilkan deteksi yang baik pada saat kondisi cuaca cerah dan pada jalan yang lurus. Namun, ketika kontur jalan memiliki tikungan, hasil deteksi dari algoritma ini tidak mampu mengikuti kelengkungan jalan sehingga menghasilkan tingkat error yang tinggi.

Di sisi lain, saat ini bidang *computer vision* saat ini berkembang pesat seiring dengan berkembangnya *deep learning*. Salah satu contoh algoritma *deep learning* adalah *Convolutional Neural Network* (CNN). CNN ini memiliki performa yang sangat baik pada permasalahan klasifikasi (Huang et al., 2017) maupun segmentasi citra (Ronneberger et al., 2015). CNN juga telah digunakan untuk permasalahan deteksi lajur dan memiliki performa yang baik. Namun, penggunaan CNN dalam deteksi lajur masih memiliki kekurangan salah satunya adalah kegagalan deteksi apabila garis lajur pudar dan terputus (Mahmud Yusuf et al., 2020). Permasalahan yang terdapat pada CNN ini dapat teratasi apabila hasil deteksi pada *timestep* sebelumnya juga dapat diproses ke dalam algoritma deteksi sehingga hasil deteksi menjadi lebih akurat. Salah satu metode dalam *deep learning* yang memiliki performa baik dalam mengolah informasi yang bersifat sekuensial dan historis ini adalah *Recurrent Neural Network* (RNN) beserta variasinya seperti *Long-Short Term Memory* (LSTM).

Kemampuan ekstraksi fitur yang baik dari CNN dan kemampuan RNN yang baik dalam mengolah informasi temporal ini memotivasi penulis untuk mengombinasikan keduanya dalam menyelesaikan permasalahan deteksi lajur pada tugas akhir ini. Hal ini dilakukan karena posisi suatu lajur pada suatu *frame* berkaitan erat dengan posisi lajur di *frame-frame* sebelumnya. Kombinasi ini diharapkan mampu menghasilkan deteksi garis lajur yang lebih akurat pada jalan raya.

1.2 Rumusan Masalah

Beberapa algoritma seperti *hough transform* (Widianto, 2020) dan *convolutional neural network* (Mahmud Yusuf et al., 2020) telah dikembangkan. Kedua algoritma ini bekerja baik ketika kondisi cuaca yang cerah dan pada jalan yang lurus. Namun, algoritma ini mengalami penurunan performa ketika menghadapi situasi tertentu seperti ketika cuaca hujan, malam, jalan yang memiliki tikungan, atau pada kondisi ketika garis lajur terdegradasi.

Salah satu metode yang dapat digunakan untuk mengatasi permasalahan ini adalah mengombinasikan antara CNN dan RNN. Kombinasi ini dilakukan untuk memanfaatkan kemampuan CNN yang baik dalam melakukan ekstraksi fitur dan kemampuan RNN yang baik dalam memproses informasi yang bersifat temporal. Sifat posisi lajur pada suatu *frame* yang berhubungan erat dengan posisi lajur pada *frame-frame* sebelumnya membuat kombinasi dari dua jenis *neural network* ini cocok untuk digunakan dalam permasalahan deteksi lajur.

Maka dari itu, permasalahan yang dapat dirumuskan pada tugas akhir ini adalah bagaimana performa dan cara mendeteksi lajur pada jalan dengan menggunakan *hybrid* CNN-RNN serta perbandingannya dengan metode deteksi lajur yang lain seperti *hough transform* yang telah dilakukan pada penelitian tugas akhir sebelumnya.

1.3 Batasan Masalah

Batasan masalah yang perlu diperhatikan dalam penelitian ini adalah sebagai berikut:

1. Kondisi jalan baik dan lajur pada jalan terlihat dengan jelas.
2. Objek yang dideteksi hanya lajur jalan. Objek lain seperti *traffic light*, kendaraan lain, dan *zebra cross* tidak ikut dideteksi
3. Dataset yang digunakan untuk proses *training* dan validasi *neural network* adalah tvtLane Dataset (Zou et al., 2019)
4. Data *testing* yang digunakan diambil secara mandiri dengan kecepatan mobil 20-30 km/jam menggunakan kamera *smartphone*, berlokasi di salah satu ruas jalan raya di Kota Malang, dan terdiri dari 3 kondisi cuaca: berawan, hujan, dan malam hari, dengan masing-masing terdiri dari 3 jenis jalan: lurus, belok kanan, belok kiri.
5. Implementasi deteksi lajur menggunakan bahasa pemrograman *Python* dengan bantuan *PyTorch* sebagai *framework* untuk *deep learning*.

1.4 Tujuan

Tujuan dari penelitian tugas akhir ini adalah menerapkan metode *hybrid* CNN-RNN dalam melakukan deteksi lajur. Penggunaan metode ini diharapkan dapat menghasilkan metrik performa dalam bentuk F1-*score* yang lebih baik dibandingkan dengan metode yang digunakan pada penelitian tugas akhir sebelumnya, yaitu *hough transform*, untuk berbagai kombinasi cuaca dan jalan.

1.5 Manfaat

Hasil penelitian ini diharapkan dapat memberikan manfaat berupa peningkatan performa deteksi lajur pada mobil otonom dibandingkan dengan performa deteksi lajur yang dihasilkan oleh penelitian sebelumnya. Penelitian ini juga diharapkan mampu untuk menjadi referensi dalam pengembangan deteksi lajur pada mobil otonom baik untuk mahasiswa Institut Teknologi Sepuluh Nopember yang akan mengambil Tugas Akhir maupun untuk peneliti lain.

BAB 2 TINJAUAN PUSTAKA

2.1 Hasil Penelitian Terdahulu

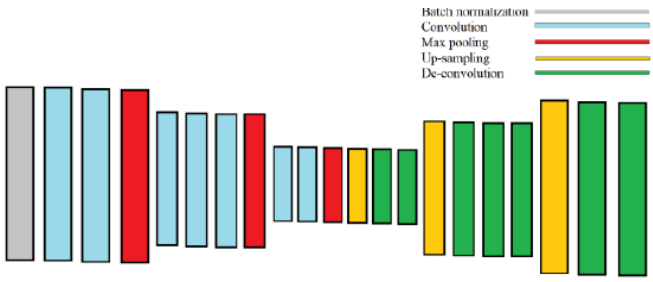
Deteksi lajur merupakan salah satu fungsi yang memainkan peran penting dalam mobil otonom. Lajur dapat didefinisikan sebagai suatu struktur garis yang kontinu, baik garis yang solid maupun garis putus-putus yang berfungsi untuk mengarahkan arus kendaraan. Persepsi tentang lajur ini dapat mencegah mobil otonom untuk mengambil posisi yang salah pada jalan.

Gambar 2.1. Contoh Hasil Deteksi Lajur

Terdapat dua pendekatan utama dalam melakukan proses deteksi lajur melalui input dari kamera, yaitu: pendekatan pertama adalah deteksi lajur berbasis *heuristic recognition* dan pendekatan ke dua adalah deteksi lajur berbasis *deep learning* (Tang et al., 2021). Pada pendekatan pertama, tahap *pre-processing* gambar masukan dan ekstraksi fitur memainkan peran penting.

Widianto, 2020 melakukan penelitian tugas akhir dengan menggunakan pendekatan pertama. Deteksi lajur dilakukan dengan melewati beberapa tahap: *greyscale transformation, gaussian blur, color mask selection*, *Canny edge detection,* menentukan ROI (*Region of Interest*), dan yang terakhir adalah *Hough line transformation*. Tahapan mulai dari *greyscale transformation* hingga penentuan ROI merupakan tahap *pre-processing* dan ekstraksi fitur. Lajur dari jalan baru didapatkan setelah *Hough line transformation* dilakukan. Algoritma ini mampu menghasilkan deteksi yang baik pada saat kondisi cuaca cerah dan pada jalan yang lurus. Namun, ketika kontur jalan memiliki tikungan, hasil deteksi dari algoritma ini tidak mampu mengikuti kelengkungan jalan sehingga menghasilkan tingkat error yang tinggi

Pendekatan kedua yang berbasis *deep learning* biasanya menggunakan metode CNN (Mahmud Yusuf et al., 2020). Pada penelitian yang telah dilakukan ini, penggunaan CNN membuat ekstraksi fitur dapat dilakukan secara otomatis dengan melatih filter-filter konvolusi untuk mempelajari fitur-fitur tertentu pada suatu gambar masukan. Arsitektur CNN yang digunakan biasanya memiliki bentuk *Encoder-Decoder* seperti gambar 2.2. Hal ini bertujuan untuk menjamin gambar keluaran dari CNN memiliki resolusi yang sama dengan gambar masukan. Namun, penggunaan CNN dalam deteksi lajur masih memiliki kekurangan salah satunya adalah kegagalan deteksi apabila garis lajur pudar dan terputus.



Gambar 2.2 Ilustrasi Arsitektur Encoder-Decoder

2.2 Dasar Teori

Tugas akhir ini mengimplementasikan beberapa teori dasar seperti mobil otonom, *deep learning, convolutional neural network,* dan *recurrent neural network*

2.2.1 Mobil Otonom

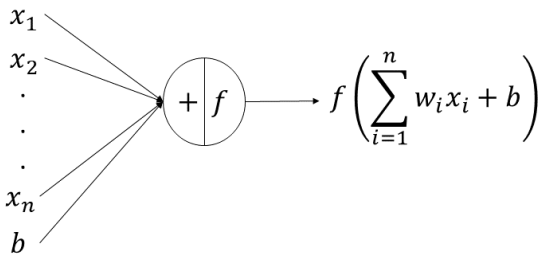
Mobil otonom merupakan suatu mobil yang mampu berjalan dan mengambil keputusan tanpa adanya intervensi manusia. Untuk menggantikan peran manusia, mobil otonom sangat bergantung pada kecerdasan buatan, sensor-sensor, dan teknologi *big data* dalam menganalisis informasi, beradaptasi ketika ada perubahan, dan mengatasi situasi kompleks di jalan raya (Taeihagh & Lim, 2019). Dengan adanya mobil otonom, berbagai manfaat dapat diperoleh seperti mengurangi tingkat kecelakaan, meningkatkan efisiensi perjalanan, dan mengurangi tingkat polusi yang ada di lingkungan.

Berdasarkan *Society of Auomotive Engineer* (SAE), tingkat keotonoman mobil otonom dapat diklasifikasikan menjadi 5 tingkatan: *assisted automation, partial automation, conditional automation, high automation,* dan *full automation* (SAE, 2018). Agar suatu mobil otonom dapat mencapai tingkatan paling tinggi, *full automation,* fungsi-fungsi dasar yang terdapat di mobil otonom harus dapat bekerja dengan baik. Menurut Jo *et. al.* (Jo et al., 2014), terdapat lima fungsi dasar dari mobil otonom, yaitu persepsi, lokalisasi, perencanaan, kontrol, dan manajemen sistem. Persepsi adalah suatu proses dari mobil otonom untuk memahami kondisi lingkungan sekitarnya dengan menggunakan beberapa sensor seperti kamera, RADAR, dan LIDAR. Proses deteksi lajur pada penelitian tugas akhir ini bisa dikategorikan sebagai bagian dari persepsi mobil otonom karena pengetahuan tentang posisi lajur dapat membantu kendaraan untuk mengetahui ke arah mana kendaraan harus berjalan dan mencegah kendaraan tersebut untuk mengambil posisi yang salah.

2.2.2 Deep Learning dan Artificial Neural Networks

Deep learning merupakan salah satu bagian dari *artificial intelligence* dan *machine learning* yang memiliki prinsip kerja dengan cara menirukan jaringan saraf manusia dalam menemukan pola atau mengolah data untuk mengambil suatu keputusan. Cara kerja jaringan saraf manusia ini terkonseptualisasi dalam suatu model dan algoritma yang disebut dengan *Artificial Neural Networks* (ANN) (J. W. G. Putra, 2020). ANN dapat menjalankan suatu tugas tanpa perlu diprogram secara ekplisit. Namun, sistem ini perlu "dilatih" terlebih dahulu agar dapat menjalankan tugas tertentu dengan baik. *Deep learning* sendiri pada dasarnya merupakan suatu proses pembelajaran untuk memahami suatu pola pada data dengan menggunakan

susunan ANN berlapis dengan neuron-neuron yang saling terhubung. Struktur dasar dari ANN digambarkan pada gambar 2.3.



Gambar 2.3 Struktur Dasar ANN

ANN memiliki tiga tahapan dalam memproses data masukannya. Tahap pertama adalah setiap nilai masukan akan dikalikan dengan bobot tertentu. Tahap kedua, dilakukan operasi penjumlahan untuk tiap masukan berbobot beserta bias, dan tahap ketiga adalah hasil penjumlahan ini kemudian akan dilewatkan ke dalam suatu fungsi aktivasi.

Model dari ANN, secara matematis, dapat dituliskan dalam persamaan 2.1 dimana y merupakan output dari ANN, x_i merupakan nilai masukan, b adalah bias, dan f adalah fungsi aktivasi.

$$y = f\left(\sum_{i=1}^{n} w_i x_i + b\right) \tag{2.1}$$

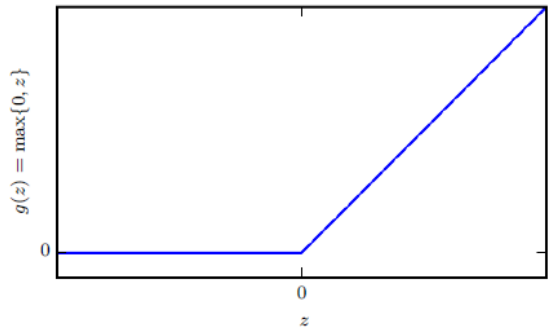
Fungsi aktivasi adalah suatu fungsi yang mengatur dan membatasi keluaran dari ANN. Secara umum, fungsi ini merupakan fungsi nonlinier. Beberapa fungsi aktivasi yang umum digunakan dalam pemodelan ANN:

a. *Rectified Linear Unit* (ReLU)

Secara matematis, fungsi ReLU dapat dideskripsikan dengan menggunakan persamaan 2.2 berikut.

$$f(x) = \max(0, x) \tag{2.2}$$

Fungsi ini akan bernilai nol apabila x kurang dari nol, dan bernilai x apabila x lebih dari dan sama dengan 0. Grafik dari funsi ReLU digambarkan pada gambar 2.4.



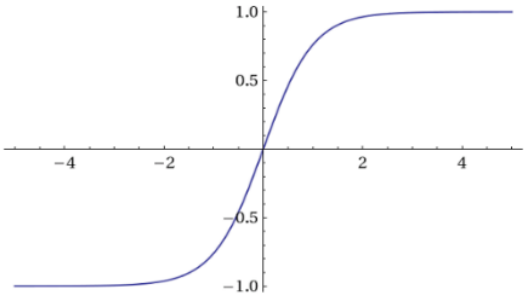
Gambar 2.4 Fungsi Rectified Linear Unit

b. *Hyperbolic Tangent* (tanh)

Secara matematis, fungsi tanh dapat dideskripsikan dengan menggunakan persamaan 2.3 berikut.

$$f(x) = \frac{e^z - e^{-z}}{e^z + e^{-z}} \tag{2.3}$$

Fungsi yang memiliki grafik menyerupai huruf 'S' ini merupakan salah satu fungsi aktivasi yang cukup sering digunakan dalam *neural network*. Fungsi tanh memiliki rentang nilai di antara -1 hingga 1. Grafik dari fungsi tanh digambarkan pada gambar 2.5.



Gambar 2.5 Fungsi Tanh

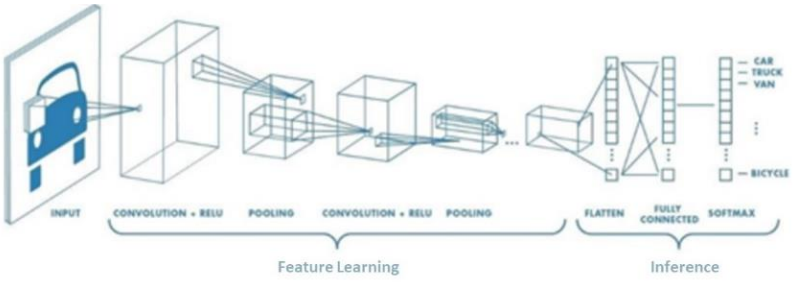
c. Softmax

Fungsi softmax adalah suatu fungsi aktivasi yang biasanya digunakan pada permasalahan klasifikasi. Fungsi ini akan menghitung probabilitas dari setiap kelas yang ada dan akan mengembalikan nilai probabilitas yang tinggi untuk kelas yang menjadi target klasifikasi. Jumlahan nilai seluruh probabilitas dari tiap kelas yang ada adalah satu. Secara matematis, softmax dapat dituliskan dengan persamaan 2.4 berikut.

$$S(y_i) = \frac{e^{y_i}}{\sum_j e^{y_j}} \tag{2.4}$$

2.2.3 Convolutional Neural Networks (CNN)

CNN merupakan salah satu jenis ANN yang umumnya digunakan untuk melakukan pengolahan data visual, baik berupa video maupun gambar. Berbeda dengan metode pengolahan citra secara konvensional yang masih memerlukan banyak *feature engineering*, CNN memiliki kemampuan yang baik dalam mengolah jenis data visual karena kemampuannya dalam mempelajari dan mengekstraksi fitur secara otomatis. Arsitektur CNN memiliki beberapa elemen dasar, yaitu *convolution layer, stride, padding, pooling,* dan *Fully Connected Layer* (Albawi et al., 2018).



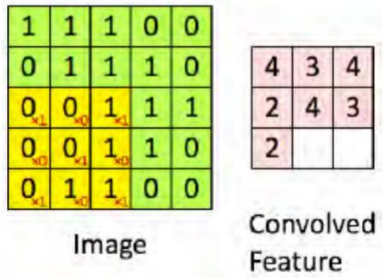
Gambar 2.6 Arsitektur Umum CNN

2.2.3.1 *Convolution Layer*

Convolution layer merupakan salah satu elemen dasar pada CNN. Pada lapisan ini terjadi proses konvolusi dari sebuah citra masukan dengan suatu filter yang memiliki ukuran panjang, lebar, dan kedalaman tertentu. Secara matematis, proses konvolusi pada suatu citra dapat dituliskan dengan menggunakan persamaan 2.5 berikut:

$$G[m,n] = (f * h)[m,n] = \sum_j \sum_k h[j,k] f[m+j, n+k] \tag{2.5}$$

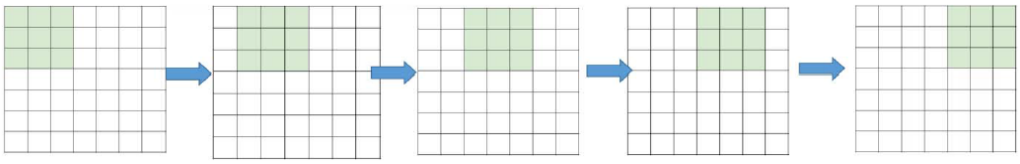
Persamaan 2.5 menjelaskan bahwa, $G[m,n]$ menyatakan *feature map* hasil konvolusi, f menyatakan citra masukan, dan h menyatakan filter atau *kernel* yang digunakan dalam proses konvolusi. Pada awalnya, nilai-nilai pada filter ini diinisialisasi dengan nilai tertentu dan kemudian akan di-update seiring berjalannya proses *learning*. Sebagai visualisasi, pada gambar 2.7, suatu filter yang berwarna kuning akan bergerak dari sudut kiri atas citra yang akan dikonvolusi (hijau) ke sudut kanan bawah citra sehingga menghasilkan gambar di bagian kanan (Putra, 2016).



Gambar 2.7 Proses Konvolusi

2.2.3.2 *Stride*

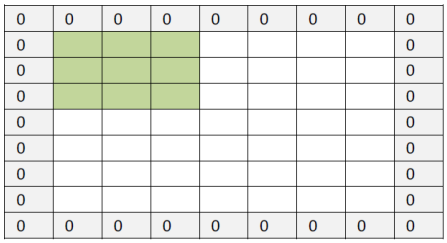
Stride merupakan banyaknya pergeseran piksel pada suatu filter saat mengalami proses konvolusi. *Stride* akan mengontrol overlap yang terjadi pada suatu citra masukan. Sebagai contoh, pada gambar 2.8, suatu citra masukan berukuran 7x7 melakukan proses konvolusi dengan matriks 3x3 dengan *stride* sebesar 1. Proses ini akan menghasilkan matriks berukuran 5x5.



Gambar 2.8 Ilustrasi *Stride* Sebesar 1

2.2.3.3 *Padding*

Operasi konvolusi biasa memiliki salah satu kelemahan, yaitu terdapat kemungkinan hilangnya informasi penting, terutama pada bagian batas atau ujung-ujung dari suatu citra masukan. Masalah ini dapat diatasi dengan suatu metode yang disebut dengan *padding*. *Padding* dilakukan dengan cara menambahkan sejumlah piksel pada citra masukan saat akan diproses oleh filter. Salah satu *padding* yang umum digunakan adalah *zero-padding*, yaitu *padding* dengan nilai piksel yang ditambahkan adalah nol.



Gambar 2.9 Proses *Zero-Padding*

Padding memainkan peran penting dalam menentukan apakah operasi konvolusi yang dilakukan adalah *valid convolution* (ukuran citra keluaran menjadi lebih kecil dari masukan) atau *same convolution* (ukuran citra keluaran sama dengan masukan) (Alsallakh et al., 2020).

Secara umum, keluaran dari citra yang telah mengalami proses konvolusi, *stride*, dan *padding* secara berurutan akan memiliki ukuran:

$$Output = \left\lfloor \frac{N + 2P - F}{S} \right\rfloor + 1 \tag{2.6}$$

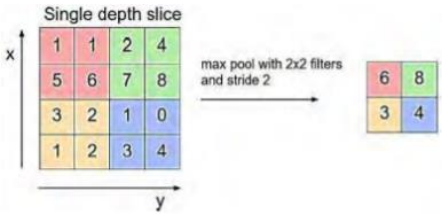
dimana N adalah ukuran citra masukan, P adalah ukuran *padding*, S adalah *stride*, dan F adalah ukuran filter yang digunakan.

2.2.3.4 *Pooling*

Pooling adalah suatu metode yang digunakan untuk mengurangi ukuran sebuah data citra (*subsampling*). Pengurangan ukuran citra ini bertujuan untuk mempercepat komputasi saat proses *learning* dari CNN. Pada umumnya, *pooling* yang digunakan oleh CNN adalah *max pooling*. *Max pooling* membagi keluaran dari lapisan konvolusi menjadi beberapa *grid* dengan ukuran tertentu, kemudian mengambil nilai maksimal dari setiap *grid* untuk membuat matriks citra yang tereduksi. Secara matematis, *max-pooling* dapat dirumuskan sebagaimana pada persamaan 2.7

$$\boldsymbol{h}_{xy}^{l} = max_{i=0,\ldots,s,j=0,\ldots,s} \boldsymbol{h}_{(x+i)(y+j)}^{l-1} \tag{2.7}$$

Sebagai ilustrasi, proses Proses *max pooling* ditunjukkan pada gambar 2.10. Selain *max pooling*, terdapat juga metode *pooling* lain seperti *sum pooling* dan *average pooling*



Gambar 2.10 *Max Pooling*

2.2.4 Recurrent Neural Network (RNN)

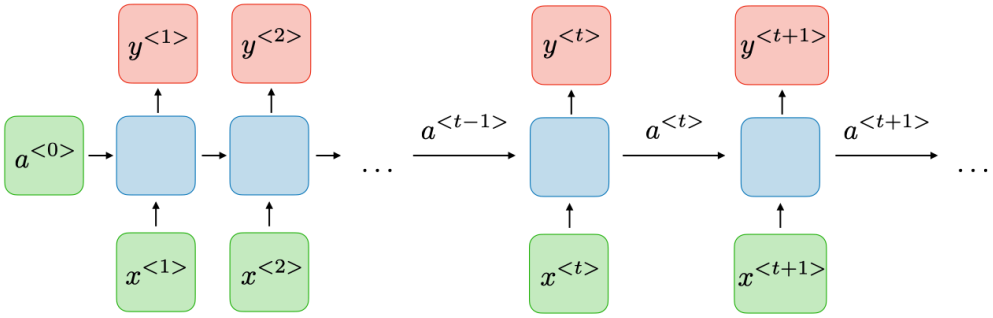
RNN merupakan salah satu jenis arsitektur ANN yang umumnya digunakan untuk mengolah data yang bersifat sekuensial atau *timeseries* (Zaremba et al., 2014). RNN memiliki kemampuan yang baik dalam mengolah jenis data ini karena RNN memiliki struktur yang mampu mengolah informasi lampau dalam proses pembelajarannya sehingga dapat mengenali pola pada data

dengan baik dan membuat prediksi yang akurat. Persamaan umum dari RNN diberikan oleh persamaan 2.8 dan 2.9:

$$a^{<t>} = g_1(W_{aa}a^{<t-1>} + W_{ax}x^{<t>} + b_a \quad (2.8)$$

$$y^{<t>} = g_2(W_{ya}a^{<t>} + b_y) \quad (2.9)$$

dimana untuk setiap *timestep* t, $a^{<t>}$ adalah aktivasi, $y^{<t>}$ adalah output, g_1 dan g_2 adalah fungsi aktivasi yang umumnya adalah fungsi *hyperbolic tangent* tanh, dan W_{ax}, W_{aa}, W_{ya}, b_a, b_y adalah *shared parameters*.

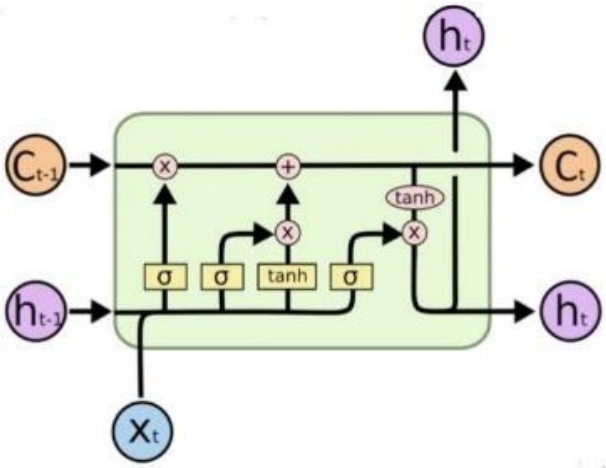


Gambar 2.11 Arsitektur RNN

Gambar 2.11 tentang arsitektur RNN menjelaskan bahwa masukan pada masa lampau $x^{<t-1>}, x^{<t-2>}, \ldots, x^{<1>}$ turut memiliki andil dalam menentukan output $y^{<t>}$. Hal inilah yang menyebabkan RNN memiliki kemampuan yang baik dalam mengolah data yang bersifat sekuensial.

2.2.4.1 *Long-Short Term Memory* (LSTM)

LSTM merupakan modifikasi dari RNN yang pertama kali dikembangkan oleh Hochreiter dan Schmidhuber (Hochreiter & Schmidhuber, 1997). LSTM dirancang untuk mengatasi permasalahan *vanishing* atau *exploding gradient* yang terjadi pada RNN saat memproses data sekuensial yang panjang. Ilustrasi dari LSTM diberikan pada gambar 2.12



Gambar 2.12 Ilustrasi LSTM

Persamaan-persamaan untuk LSTM dideskripsikan oleh persamaan 2.10 sampai 2.15

$$f_t = \sigma(W_f[h_{t-1}, x_t] + b_f) \tag{2.10}$$

$$i_t = \sigma(W_i[h_{t-1}, x_t] + b_i) \tag{2.11}$$

$$\tilde{C}_t = \tanh\,(W_C[h_{t-1}, x_t] + b_C) \tag{2.12}$$

$$C_t = f_t \circ C_{t-1} + i_t \circ \tilde{C}_t \tag{2.13}$$

$$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o) \tag{2.14}$$

$$h_t = o_t \circ \tanh(C_t) \tag{2.15}$$

dimana f_t adalah *forget gate*, i_t adalah *input gate*, C_t adalah *cell state*, o_t adalah *output gate*, h_t adalah *hidden state*, dan $\circ$ menunjukkan *Hadamard* atau *element-wise product*.

Tahap pertama dari suatu LSTM *cell* adalah menentukan informasi apa yang akan dibuang pada suatu *cell state*. Keputusan penyimpanan informasi ini ditentukan oleh *forget gate*. Nilai keluaran dari *forget gate* berada di antara nol, yang berarti melupakan secara total informasi pada *cell state*, dan satu yang berarti menyimpan keseluruhan informasi pada *cell state*. Tahap berikutnya adalah menentukan informasi baru apa yang akan disimpan di *cell state*. Tahap kedua ini memiliki dua bagian utama yaitu lapisan sigmoid yang bekerja sebagai *input gate* dan lapisan tanh yang akan membentuk suatu vektor kandidat pengganti C_t yaitu $\tilde{C}_t$.

Keluaran dari *input gate* dan layer tanh ini kemudian akan digabungkan untuk melakukan *update* pada *cell state* yang lama, C_{t-1}, seperti yang tertera pada persamaan 2.13. Tahap terakhir adalah menentukan apa yang akan menjadi keluaran dari LSTM *cell*. Keluaran ini merupakan *cell state* yang mengalami filtering oleh *output gate*. Pada tahap terakhir ini, nilai dari *cell state* akan dilewatkan ke fungsi tanh dan akan dikalikan dengan keluaran dari output gate seperti yang tertera pada persamaan 2.15.

2.2.5 Training Neural Network

Suatu model *neural network* harus dilatih (*training*) agar dapat menyelesaikan objektif tertentu seperti regresi, klasifikasi, maupun segmentasi. Tahapan roses *training* dari *neural network* dijabarkan pada subbab 2.2.5.1 sampai 2.2.5.3 berikut.

2.2.5.1 *Forward Propagation*

Tahap pertama dari proses *training neural network* adalah *forward propagation*. Pada proses ini, parameter-parameter yang ada di model *neural network* pada mulanya akan diinisialisasi secara acak. Kemudian, suatu data yang berada dalam dataset *training* akan dimasukkan ke model *neural network* sebagai *input* $\boldsymbol{x}$. *Input* $\boldsymbol{x}$ ini memberikan informasi awal yang kemudian akan dipropagasikan ke seluruh *hidden units* pada setiap lapisan. Setelah melewati seluruh *hidden unit* yang ada, suatu output $\hat{\boldsymbol{y}}$ akan dihasilkan. Keseluruhan proses dari $\boldsymbol{x}$ menjadi $\hat{\boldsymbol{y}}$ inilah yang disebut dengan *forward propagation* (Goodfellow et al., 2016).

2.2.5.2 Perhitungan *Cost*

Setelah suatu model *neural network* menghasilkan *output* $\hat{\boldsymbol{y}}$, dilakukan pengomparasian dengan label atau *ground truth*. Komparasi ini dilakukan dengan menggunakan fungsi yang

disebut dengan *cost function*, suatu fungsi bernilai skalar yang mengukur seberapa baik keseluruhan model *neural network* dalam menghasilkan output $\hat{\boldsymbol{y}}$ jika diberi masukan $\boldsymbol{x}$ dibandingkan dengan *ground truth* $\boldsymbol{y}$. Beberapa *cost function* yang umum digunakan dalam proses *training neural network*:

a. Mean Absolute Error (MAE)

MAE merupakan metrik yang umumnya digunakan dalam permasalahan regresi untuk mengukur rata-rata selisih mutlak antara nilai *ground truth* ($\boldsymbol{y}$) dengan nilai prediksi dari model *neural network* ($\hat{\boldsymbol{y}}$). Secara matematis MAE dapat dirumuskan pada persamaan 2.16.

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \tag{2.16}$$

b. Mean Squared Error (MSE)

MSE adalah rata-rata kesalahan kuadrat antara nilai *ground truth* dan nilai prediksi dari model *neural network*. Secara matematis, MSE dapat dirumuskan pada persamaan 2.17.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{2.17}$$

c. Binary Cross-Entropy (BCE)

BCE pada umumnya digunakan pada permasalahan klasifikasi biner yang memiliki nilai target 0 atau 1. Metrik ini akan menghitung perbedaan antara distribusi probabilitas hasil prediksi dengan nilai aktual dalam memprediksi kelas 1. Semakin baik hasil prediksi dari model *neural network*, nilai BCE *loss* akan mendekati nilai 0. Secara matematis, BCE dapat dirumuskan pada persamaan 2.18

$$BCE = -\frac{1}{n}\sum_{i=1}^{n} y_i \log \hat{y}_i + (1 - y_i)\log\,(1 - \hat{y}_i) \tag{2.18}$$

2.2.5.3 *Backpropagation*

Tahap berikutnya setelah perhitungan *cost* dilakukan adalah *backpropagation*. *Backpropagation* bertujuan untuk meminimalisasi *cost function* dengan melakukan *update* dan penyesuaian pada parameter-parameter di *neural network* seperti *weights* dan *biases*. Besar penyesuaian parameter-parameter ini ditentukan oleh besar gradien *cost function* terhadap parameter-parameter tersebut. Berikut adalah beberapa algoritma yang umum digunakan untuk meng-*update* nilai parameter-parameter sehingga dapat meminimalisasi *cost function*:

a. *Stochastic Gradient Descent* (SGD)

SGD merupakan algoritma yang dapat digunakan untuk melakukan *update* parameter-parameter yang ada di *neural network* seperti *weight* dan bias. Tujuan *update* parameter ini adalah meminimalisasi nilai dari *cost function*. SGD melakukan *update* parameter dengan cara mengurangi *weight* dan bias inisial dengan sebagian nilai gradien *cost function* terhadap parameter yang didapat.

Parameter yang menentukan seberapa banyak gradien yang digunakan dalam meng-*update* parameter *neural network* disebut dengan *learning rate*. Nilai dari *learning rate* ini memiliki rentang antara nol dan satu. Apabila *learning rate* bernilai mendekati nol, parameter model *neural network* akan diperbarui secara perlahan. Sebaliknya, apabila mendekati satu, maka

parameter di model akan diperbarui secara cepat. Secara matematis, SGD dapat dituliskan dengan persamaan 2.19 dan 2.20.

$$W_{j+1} = w_j - \alpha \frac{\partial}{\partial_{w_j}} J(W, b) \tag{2.19}$$

$$b_{j+1} = b_j - \alpha \frac{\partial}{\partial_{b_j}} J(W, b) \tag{2.20}$$

Pada persamaan 2.18 dan 2.19, W_{j+1} dan b_{j+1} menyatakan *weight* dan bias pada iterasi selanjutnya, w_j dan b_j menyatakan weight dan bias pada iterasi saat ini, α menyatakan *learning rate* dan $J(W, b)$ menyatakan *cost function*.

b. *Adaptive Moment Estimation* (Adam)

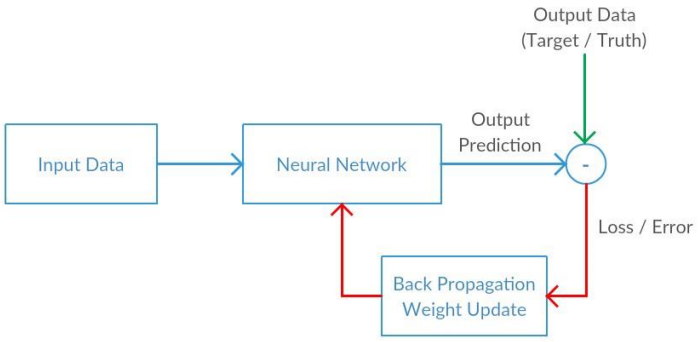
Adam merupakan salah satu algoritma optimisasi yang dapat digunankan dalam proses *training* untuk meng-*update* parameter-parameter dari *neural network*. Pada prosesnya, *learning rate* dari Adam dapat berubah secara adaptif pada saat proses *training*. Secara matematis, Adam dapat dituliskan dengan persamaan 2.21 sampai 2.23.

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) \frac{\partial}{\partial_{w_t}} J(w, b) \tag{2.21}$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) \left(\frac{\partial}{\partial_{w_t}} J(w, b) \right)^2 \tag{2.22}$$

$$\theta_j = \theta_{j-1} - \frac{\alpha}{\sqrt{v_t + \varepsilon}} m_t \tag{2.23}$$

Keseluruhan proses mulai dari *forward propagation*, perhitungan *cost*, dan *backpropagation* terus diulangi hingga nilai *cost function* menjadi seminimal mungkin. Proses *training* dari model *neural network* ini diilustrasikan pada gambar 2.13.



Gambar 2.13 Proses *Training Neural Network*

2.2.6 *Hybrid* CNN-RNN untuk Deteksi Lajur

Deteksi lajur dengan menggunakan CNN biasanya dilakukan pada *current frame* suatu masukan, sehingga informasi-informasi yang berasal dari beberapa *frame* sebelumnya tidak diperhitungkan untuk melakukan deteksi. Hal ini meningkatkan kemungkinan terjadinya penurunan kualitas deteksi lajur pada jalan. Suatu arsitektur yang mampu mengolah informasi

temporal diperlukan karena posisi lajur dari frame yang berdekatan berhubungan erat dan lajur pada suatu *current frame* dapat diprediksi menggunakan beberapa *frame* sebelumnya.

RNN merupakan jenis *neural network* yang memiliki kemampuan baik dalam mengolah informasi temporal. Dengan demikian, kombinasi CNN dan RNN dapat menjadi suatu metode untuk digunakan dalam deteksi lajur. Arsitektur yang mengkombinasikan dua jenis *neural network* ini adalah UNet-ConvLSTM dan SegNet-ConvLSTM (Zou et al., 2019). Dengan menggunakan arsitektur ini, deteksi lajur akan dilakukan dengan cara *semantic segmentation* untuk mengenali *pixel-pixel* gambar masukan yang termasuk dalam suatu lajur jalan.

2.2.6.1 SegNet-ConvLSTM

Arsitektur SegNet-ConvLSTM merupakan kombinasi dari SegNet (Badrinarayanan et al., 2015) dan ConvLSTM (Shi et al., 2015). ConvLSTM merupakan salah satu variasi dari LSTM dimana proses perkalian matriks pada setiap *gate* di LSTM digantikan dengan proses konvolusi. Maka dari itu, persamaan 2.10 hingga 2.15 berubah menjadi persamaan 2.24 hingga 2.29.

$$f_t = \sigma(W_{xf} * x_t + W_{hf} * h_{t-1} + W_{cf} \circ C_{t-1} + b_f) \quad (2.24)$$

$$i_t = \sigma(W_{xi} * x_t + W_{hi} * h_{t-1} + W_{ci} \circ C_{t-1} + b_i) \quad (2.25)$$

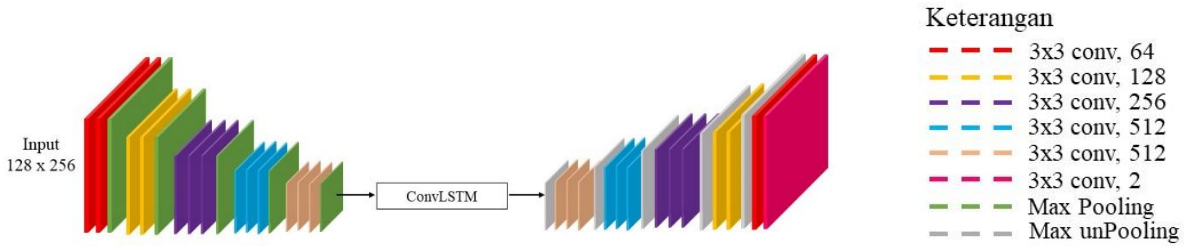
$$\tilde{C_t} = \tanh\,(W_{xc} * x_t + W_{hc} * h_{t-1} + b_c) \quad (2.26)$$

$$C_t = f_t \circ C_{t-1} + i_t \circ \tilde{C_t} \quad (2.27)$$

$$o_t = \sigma(W_{xo} * x_t + W_{ho} * h_{t-1} + W_{co} \circ C_t + b_o) \quad (2.28)$$

$$h_t = o_t * \tanh(C_t) \quad (2.29)$$

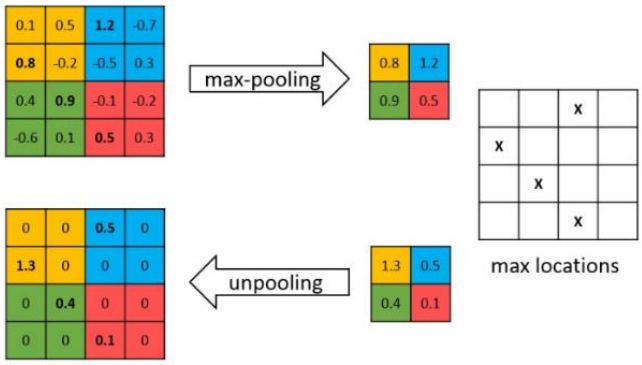
Terdapat tiga blok utama dari SegNet-ConvLSTM: *encoder*, ConvLSTM, dan *decoder*. Pada bagian *encoder*, terjadi proses ekstraksi fitur mulai dari fitur yang sederhana seperti *edge* pada suatu gambar hingga fitur-fitur kompleks yang merupakan kombinasi dari fitur-fitur sederhana. Gambar 2.14 menunjukkan keseluruhan arsitektur beserta keterangannya dari SegNet-ConvLSTM.



Gambar 2.14 Arsitektur SegNet-ConvLSTM

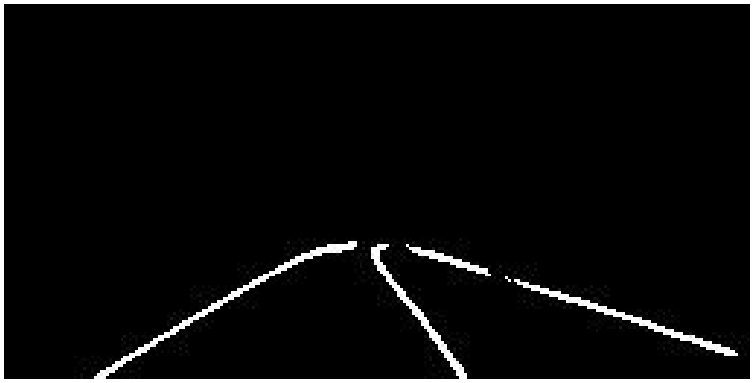
Terdapat beberapa tahapan yang dilalui oleh citra masukan sebelum menghasilkan deteksi lajur. Pertama, citra masukan akan di-*resize* menjadi ukuran 128 x 256. Kemudian citra akan mengalami *downsampling* dengan cara melewati 5 (lima) blok *convolution-pooling* seperti yang terlihat pada gambar 2.14. *Pooling* yang digunakan pada bagian *encoder* adalah *Max-Pooling*. Pada saat proses *pooling*, index dari nilai piksel yang maksimum akan disimpan untuk proses *upsampling* di bagian *decoder*. Setelah *feature map* yang dihasilkan memiliki ukuran 4

x 8 x 512, *feature map* akan masuk ke blok ConvLSTM untuk pengolahan informasi temporal. Selanjutnya, *feature map* akan masuk ke bagian *decoder* dengan tujuan mengembalikan ukuran *feature map* menjadi sama dengan ukuran masukan. Fungsi aktivasi yang digunakan pada tiap *layer* adalah ReLU. Pada bagian *decoder*, terjadi proses *upsampling* menggunakan metode *max-unpooling*. Gambar 2.15 mengilustrasikan proses *max-unpooling*.



Gambar 2.15 Proses *Max-Unpooling*

Setelah melewati beberapa konvolusi dan *max-unpooling*, resolusi *feature map* akan memiliki resolusi yang sama dengan masukan. Selain itu, citra yang dihasilkan akan memiliki dua buah *channel*: *background* dan lajur yang terdeteksi. Gambar 2.16 menunjukkan contoh hasil deteksi lajur yang didapatkan.



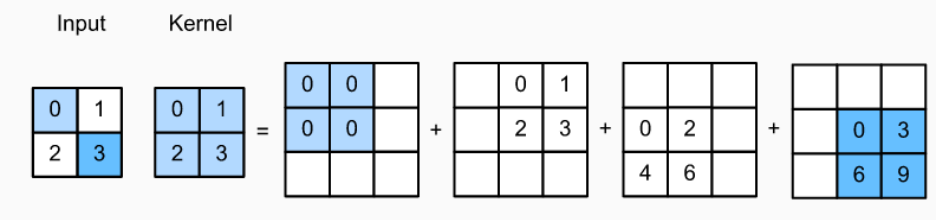
Gambar 2.16 Contoh Hasil Deteksi Lajur

2.2.6.2 UNet-ConvLSTM

Arsitektur UNet-ConvLSTM merupakan kombinasi dari UNet (Ronneberger et al., 2015) dan ConvLSTM. Seperti pada SegNet-ConvLSTM, arsitektur ini secara garis besar juga memiliki tiga blok utama: *encoder*, ConvLSTM, dan *decoder*. Arsitektur UNet-ConvLSTM memiliki beberapa perbedaan dengan SegNet-ConvLSTM. Pertama, pada UNet-ConvLSTM, masing-masing filter terjadi dua konvolusi secara berurutan. Sedangkan pada SegNet-ConvLSTM, terjadi tiga konvolusi berurutan pada ukuran filter 3x3x256 dan 3x3x512.

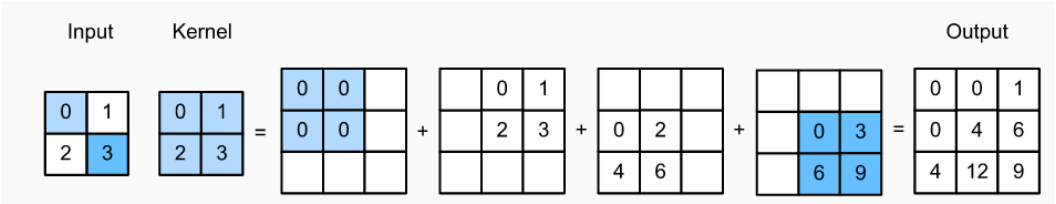
Perbedaan kedua adalah UNet-ConvLSTM menggunakan *skip-connection* untuk melakukan konkatenasi antara *feature map* yang dihasilkan pada blok *encoder* dan *feature map* di blok *decoder*. Hal ini bertujuan untuk memperkaya informasi spasial sehingga lajur yang terdeteksi menjadi semakin akurat.

Perbedaan yang ketiga adalah metode *upsampling* yang digunakan pada UNet-ConvLSTM adalah *transpose convolution*. Metode ini bertujuan untuk mengembalikan *feature map* ke citra yang memiliki ukuran resolusi yang sama dengan gambar input. Misalkan terdapat suatu *feature map* dengan ukuran 2x2 yang ingin di *upsampling* menjadi 3x3. Suatu filter dengan ukuran 2x2 dengan *stride* sebesar 1 dan *zero padding* digunakan. Kalikan elemen paling kiri atas dari *feature map* dengan setiap elemen pada filter. Lakukan hal yang sama pada elemen-elemen yang lain sebagaimana terlihat pada gambar 2.17.



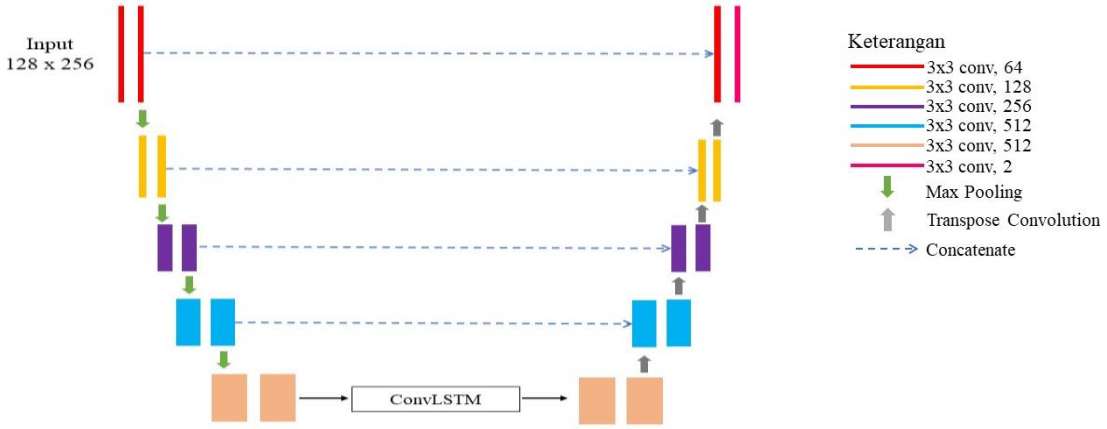
Gambar 2.17 Proses *Transpose Convolution*

Gambar 2.17 memperlihatkan bahwa beberapa elemen yang dihasilkan dari proses *upsampling* saling *overlap*. Untuk mengatasi hal ini, secara sederhana, bisa menambahkan nilai-nilai elemen yang *overlap* ini. Gambar 2.18 menunjukkan output yang dihasilkan dari contoh operasi *transpose convolution* ini.



Gambar 2.18 *Output* Proses *Transpose Convolution*.

Proses selanjutnya, lapisan konvolusi beserta proses-proses seperti *skip-connection* dan *transpose convolution* digabungkan sedemikian rupa sehingga membentuk arsitektur UNet-ConvLSTM seperti yang terlihat pada gambar 2.19. Fungsi aktivasi yang digunakan pada tiap *layer* adalah ReLU. Citra keluaran yang dihasilkan kemudian akan memiliki resolusi yang sama dengan citra masukan dengan dua buah *channel*: *background* dan lajur yang terdeteksi.



Gambar 2.19 Arsitektur UNet-ConvLSTM

2.2.7 Metrik Performa Algoritma

Kinerja dari sebuah model *deep learning*, khususnya dalam permasalahan yang bersifat *binary*, dapat diukur dengan suatu metrik performa yang disebut dengan *confusion matrix*. *Confusion matrix* memiliki 4 jenis data: *True Positive* (TP), *True Negative* (TN), *False Positive* (FP), dan *False Negative* (FN). Dalam kasus deteksi lajur pada jalan, *confusion matrix* yang digunakan memiliki struktur seperti pada tabel 2.1.

Tabel 2.1. *Confusion Matrix* Pada Kasus Deteksi Lajur

		Ground Truth	
		Terdapat Lajur	Tidak terdapat lajur
Hasil Deteksi	Terdeteksi Lajur (*Positive*)	*True Positive* (TP)	*False Positive* (FP)
	Tidak terdeteksi lajur (*Negative*)	*False Negative* (FN)	*True Negative* (TN)

Pixel lajur yang terdeteksi sebagai lajur merupakan *true positive* (TP). *Pixel* lajur yang tidak terdeteksi sebagai lajur merupakan *false negative* (FN). *Pixel* bukan lajur yang terdeteksi sebagai lajur merupakan *false positive* (FP). *Pixel* yang bukan lajur dan terdeteksi sebagai bukan lajur merupakan *true negative* (TN).

Empat nilai pada tabel 2.1, yaitu TP, FN, FP, dan TN dapat digunakan untuk menghitung tingkat akurasi, presisi, *recall* dari suatu model *deep learning*. Tingkat akurasi, presisi, *recall* tersebut memiliki jangkauan nilai 0 sampai 1 dan digunakan untuk mengukur performa dari suatu model. Persamaan 2.30 hingga 2.32 merupakan persamaan yang digunakan untuk menghitung Tingkat akurasi, presisi, *recall*.

$$akurasi = \frac{TP + TN}{TP + TN + FP + FN} \tag{2.30}$$

$$presisi = \frac{TP}{TP + FP} \tag{2.31}$$

$$recall = \frac{TP}{TP + FN} \tag{2.32}$$

Tingkat akurasi menyatakan seberapa banyak hasil prediksi yang benar (*true positive* dan *true negative*) dari keseluruhan data. Tingkat presisi menyatakan tingkat ketelitian dari suatu model *deep learning* untuk melakukan deteksi secara benar (*true positive*) pada data kategori positif yang diperoleh (*true positive* dan *false positive*). Tingkat presisi yang semakin tinggi menunjukkan bahwa kesalahan prediksi dimana kondisi sebenarnya tidak terdapat lajur semakin kecil. Tingkat *recall* menunjukkan seberapa baik model *deep learning* dapat memprediksi kategori positif (*true* positive) dari keseluruhan target positif sebenarnya (*true positive* dan *false negative*). Semakin tinggi nilai *recall*, kemungkinan kegagalan dalam deteksi dimana kondisi sebenarnya terdapat lajur semakin kecil.

Tingkat *false positive* dan *false negative* dari suatu model berperan penting dalam permasalahan deteksi lajur. Kejadian seperti kesalahan deteksi, yakni ketika terdapat lajur namun tidak dapat terdeteksi maupun sebaliknya, maka hal ini harus diminimalisasi. Kondisi ini menimbulkan kebutuhan adanya metrik yang menyeimbangkan antara nilai presisi dan

recall. Metrik tersebut adalah F_1 *score*, yaitu *harmonic mean* antara nilai *recall* dan presisi (Dalianis, 2018). Nilai F_1 *score* dapat dihitung dengan menggunakan persamaan 2.33

$$F_1 = \frac{2TP}{2TP + FP + FN} \tag{2.33}$$

Halaman ini sengaja dikosongkan

BAB 3 METODOLOGI

3.1 Urutan Pelaksanaan Penelitian

Penelitian tugas akhir ini dilaksanakan dengan urutan sebagai berikut:

1. Studi Literatur
2. Pengumpulan Dataset
3. Pembuatan Model *Hybrid* CNN-RNN
4. Desain Eksperimen dan Simulasi
5. Pengujian Algoritma dan Evaluasi
6. Penyusunan Laporan

Keenam tahap di atas dijabarkan pada subbab 3.1.1 hingga subbab 3.1.6

3.1.1 Studi Literatur

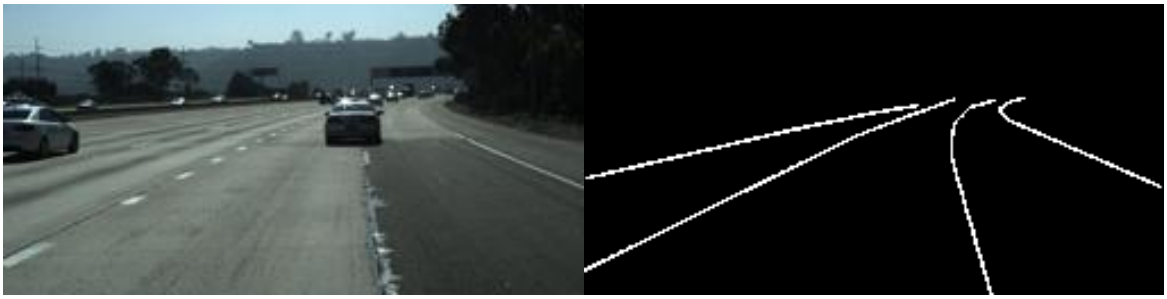
Studi literatur dilakukan untuk mencari teori dan referensi terkait permasalahan yang akan diselesaikan dalam tugas akhir ini. Beberapa topik yang dipelajari dalam proses studi literatur ini adalah Deteksi Lajur, *Convolutional Neural Network*, dan *Recurrent Neural Network*. Sumber yang digunakan untuk studi literatur ini berasal dari buku, jurnal dan *proceeding* Ilmiah, serta laporan tugas akhir terdahulu.

3.1.2 Pengumpulan Dataset

Dataset yang digunakan untuk melatih arsitektur *neural network* pada tugas akhir ini adalah tvtLane (Zou et al., 2019). Dataset ini merupakan pengembangan dari dataset TuSimple. Beberapa augmentasi citra dilakukan untuk menambah tingkat variasi pada data. Penggunaan dataset yang beragam ini diharapkan pemanfaatan model *neural network* menjadi lebih akurat.

Dataset yang digunakan terdiri dari 19383 *image sequences* untuk deteksi lajur dan sebanyak 39460 *frame* diantaranya telah diberi label. *Ground truth* dari data juga telah diberikan, sehingga tidak perlu dilakukan segmentasi secara manual untuk proses *training*.

Dataset ini dibagi menjadi dua bagian utama, yaitu dataset untuk *training* dan dataset untuk *validation*. Dataset yang digunakan untuk *training* terdiri dari 9548 gambar berlabel yang telah diaugmentasi berupa rotasi dan *flipping*. Dataset yang digunakan untuk *validation* terdiri dari 1268 gambar berlabel. Ukuran gambar dari dataset yang kemudian akan menjadi data masukan di model *neural network* adalah 128 x 256. Gambar 3.1 menunjukkan salah satu contoh gambar pada dataset beserta *ground truth*-nya.



Gambar 3.1 Contoh Gambar Pada Dataset *Training* Beserta *Ground Truth*-nya

3.1.3 Pembuatan Model *Hybrid* CNN-RNN untuk Deteksi Lajur

Pembuatan model *Hybrid* CNN-RNN yang akan digunakan untuk deteksi lajur dilakukan dengan menggunakan bahasa python dan *framework* PyTorch. Pembuatan dua model *Hybrid* CNN-RNN berupa SegNet-ConvLSTM dan UNet-ConvLSTM ini mengacu pada arsitektur yang telah dijelaskan sub-bab 2.2.6.1 dan 2.2.6.2. Sebelum menginisiasi model SegNet-ConvLSTM dan UNet-ConvLSTM, diperlukan beberapa *function* yang akan digunakan dalam pembuatan model, seperti ConvLayer, Max-Pooling, *Batch Normalization*, dan *Transposed Convolution*. Masing-masing fungsi ini ditunjukkan oleh Algoritma 1 sampai dengan Algoritma 4 berikut.

ALGORITMA 1: ConvLayer

Function: Conv2d(*in_ch, out_ch, kernel size, padding*)
Input: *Image* $\boldsymbol{I} \in R^{H \times W \times C}$
Initialization: *Kernel* $\boldsymbol{K} \in R^{k_1 \times k_2 \times C \times D}$, *biases* $\boldsymbol{b} \in R^{D}$
Output*: Convolved Feature Maps*
for *m* ***in*** *range* $(0, k_1 - 1)$*:*
    for *n* ***in*** *range* $(0, k_2 - 1)$:
        for *c* ***in*** *range* $(1, C)$:
            $\boldsymbol{K}_{m,n,c} \cdot \boldsymbol{I}_{i+m,j+n,c} + \boldsymbol{b}$
        End for
    End for
End for

ALGORITMA 2: Batch Normalization

Function *BatchNorm2d(num_features)*
Input: *Values of x over a mini-batch:* $\beta = \{x_1, \dots, m\}$;
    Parameters to be learned: γ, β
Output*:* $\{y_i = BN_{\gamma,\beta}(x_i)\}$
$\mu_\beta \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i$ // ***mini-batch mean***
$\sigma_\beta^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_\beta)^2$ // ***mini-batch variance***
$\widehat{x_i} \leftarrow \frac{x_i - \mu_\beta}{\sqrt{\sigma_\beta^2 + \epsilon}}$ // ***normalize***
$y_i \leftarrow \gamma \widehat{x_i} + \beta \equiv BN_{\gamma,\beta}(x_i)$ // ***scale and shift***

ALGORITMA 3: Max-Pooling

Function: *MaxPool2d(Image Input, kernel_size, padding)*
Input: *Parameters pool size in stride*
Output*: Feature maps with reduced dimension*
for *all rows* ***i*** *with the step size=stride:*
    for *all columns* ***j*** *the step size=stride:*
        out = original_featureMap[i : i+pool_size, j : j+pool_size]
    end for
end for

ALGORITMA 4: Transposed Convolution

Function: *ConvTranspose2d (in_ch,out_ch)*
Input: *Input Matrix X, Kernel Matrix K*

***Output**: Upsampled feature map*
h, w = K.shape
Y = zeros((X.shape[0] + h - 1, X.shape[1] + w - 1))
***for** i **in** range (0, number of row):*
    ***for** j **in** range(0, number of columns):*
        *Y[i: i + h, j: j + w] += X[i, j] * K*
    End for
End for

a. SegNet-ConvLSTM

Sub-bab 2.2.6.1 menjelaskan bahwa terdapat tiga bagian utama pada SegNet-ConvLSTM: *encoder*, ConvLSTM, dan *decoder*. Bagian *encoder* dari SegNet-ConvLSTM diinisiasi dengan *weights* yang terdapat pada model VGG16. Hal ini bertujuan untuk menghindari inisiasi *weights* dengan nilai acak sehingga dapat mempercepat proses *training* dari arsitektur *neural network* ini. Berikut adalah *pseudocode* untuk menginisiasi bagian *encoder* dari arsitektur ini.

ALGORITMA 5: Function Encoder SegNet

***Input:** Frame in driving scene video with size of 128 × 256*
***Initialization:** Weights and bias from VGG16 Architecture*
***Output**: Feature map with size of 4×8×512*
function Encoder(input)
    Conv2d(in_ch=3, out_ch=64, kernel = 3x3, padding=1)
    Conv2d(in_ch=64, out_ch=64, kernel = 3x3, padding=1)
    Conv2d(in_ch=64, out_ch=64, kernel = 3x3, padding=1)
    MaxPool2d(kernel_size = 2x2)
    Conv2d(in_ch=64, out_ch=128, kernel = 3x3, padding=1)
    Conv2d(in_ch=128, out_ch=128, kernel = 3x3, padding=1)
    MaxPool2d(kernel_size = 2x2)
    Conv2d(in_ch=128, out_ch=256, kernel = 3x3, padding=1)
    Conv2d(in_ch=256, out_ch=256, kernel = 3x3, padding=1)
    Conv2d(in_ch=256, out_ch=256, kernel = 3x3, padding=1)
    MaxPool2d(kernel_size = 2x2)
    Conv2d(in_ch=256, out_ch=512, kernel = 3x3, padding=1)
    Conv2d(in_ch=512, out_ch=512, kernel = 3x3, padding=1)
    Conv2d(in_ch=512, out_ch=512, kernel = 3x3, padding=1)
    ***return** Feature Map 4x8x512*
End function

Setelah melakukan pembuatan *encoder*, selanjutnya adalah melakukan pembuatan model ConvLSTM. Berikut adalah *pseudocode* yang digunakan untuk membuat model ConvLSTM.

ALGORITMA 6: Function Convolutional LSTM

***Input:** Feature map from previous encoder*
***Initialization:** hidden state dimension, kernel size, randomly initiate weight and bias*
***Output**: Feature map with size of 4×8×512*
***Function forward_pass**(input_tensor, cur_state):*
    h_cur, c_cur = current_state
    combined = concatenate (input tensor, h_cur)
    Conv2d(combined), 3x3 kernel, 4x8 input_size, 512x512 hidden_dim
    cci, ccf, cco, ccg = split(ConvLayer(combined))
    i = sigmoid(cci)
    f = sigmoid(ccf)
    o = sigmoid(cco)

```
        g = tanh(ccg)
        c_next = f * c_cur + i * g
        h_next = o * tanh(c_next)
        return h_next, c_next
End function
```

Selanjutnya, dilakukan pembuatan untuk *decoder* dari SegNet-ConvLSTM. Berikut adalah *pseudocode* yang digunakan untuk membuat model *decoder* untuk SegNet-ConvLSTM.

ALGORITMA 7: Function Decoder SegNet

```
    Input: Feature map with size of 4x8x512
    Initialization: Randomly initiate weight and bias of convolution filters
    Output: 2-channels image: Predicted Background and Lane
    Function Decoder (Input):
        MaximumUnpooling(Kernel_size=2)
        Conv2d(in_ch=512, out_ch=512, kernel = 3x3, padding=1)
        BatchNorm2d(512)
        Conv2d(in_ch=512, out_ch=512, kernel = 3x3, padding=1)
        BatchNorm2d(512)
        Conv2d(in_ch=512, out_ch=512, kernel = 3x3, padding=1)
        Layer1 = BatchNorm2d(512)
        MaximumUnpooling(Layer 1, kernel_size=2)
        Conv2d(in_ch=512, out_ch=512, kernel = 3x3, padding=1)
        BatchNorm2d(512)
        Conv2d(in_ch=512, out_ch=512, kernel = 3x3, padding=1)
        BatchNorm2d(512)
        Conv2d(in_ch=512, out_ch=256, kernel = 3x3, padding=1)
        Layer2 = BatchNorm2d(256)
        MaximumUnpooling(Layer 2,kernel_size=2)
        Conv2d(in_ch=256, out_ch=256, kernel = 3x3, padding=1)
        BatchNorm2d(256)
        Conv2d(in_ch=256, out_ch=256, kernel = 3x3, padding=1)
        BatchNorm2d(256)
        Conv2d(in_ch=256, out_ch=128, kernel = 3x3, padding=1)
        Layer3 =  BatchNorm2d(128)
        MaximumUnpooling(Layer 3, kernel_size=2)
        Conv2d(in_ch=128, out_ch=64, kernel = 3x3, padding=1)
        BatchNorm2d(64)
        Conv2d(in_ch=64, out_ch=64, kernel = 3x3, padding=1)
        Layer 4 = BatchNorm2d(64)
        MaximumUnpooling(Layer4, kernel_size = 2)
        Conv2d(in_ch=64, out_ch=64, kernel = 3x3, padding=1)
        BatchNorm2d(64)
        Conv2d(in_ch=64, out_ch=2, kernel = 3x3, padding=1)
        Output
    End function
```

Terakhir, ketiga bagian di atas digabungkan menjadi arsitektur SegNet-ConvLSTM yang utuh. Berikut adalah *pseudocode* untuk menggabungkan ketiga bagian tersebut.

ALGORITMA 8: Arsitektur SegNet-ConvLSTM

```
    Input: Frame from driving scene video with size of 128x256
    Initialization: Randomly initiate weight and bias of convolution filters
    Output: 2-channels image: Predicted Background and Lane
```

```
listOfFrame = [ ]
for frame in FiveConsecutiveFrames:
    Enc = function Encoder SegNet (frame)
    listOfFrame.append(Enc)
ConvLSTM = function ConvLSTM (listOfFrame)
Dec = function Decoder SegNet (ConvLSTM)
Output 2-channels image
End
```

b. UNet-ConvLSTM

Arsitektur ini juga terdiri dari tiga bagian utama: *encoder*, ConvLSTM, dan *decoder*. Pada arsitektur UNet-ConvLSTM, masing-masing filter pada bagian *encoder* maupun *decoder* terjadi dua proses konvolusi secara berurutan. Berikut adalah *pseudocode* untuk menginisiasi *encoder* dari UNet-ConvLSTM

ALGORITMA 9: Function Encoder UNet

```
Input: Frame in driving scene video with size of 128 × 256
Initialization: Randomly initiate weights and biases
Output: Feature map with size of 8x16x512
function Encoder(input)
    Conv2d(in_ch=64, out_ch=64, kernel = 3x3, padding=1)
    Conv2d(in_ch=64, out_ch=128, kernel = 3x3, padding=1)
    MaxPool2d(kernel_size = 2x2)
    Conv2d(in_ch=128, out_ch=128, kernel = 3x3, padding=1)
    Conv2d(in_ch=128, out_ch=256, kernel = 3x3, padding=1)
    MaxPool2d(kernel_size = 2x2)
    Conv2d(in_ch=256, out_ch=256, kernel = 3x3, padding=1)
    Conv2d(in_ch=256, out_ch=512, kernel = 3x3, padding=1)
    MaxPool2d(kernel_size = 2x2)
    Conv2d(in_ch=512, out_ch=512, kernel = 3x3, padding=1)
    Conv2d(in_ch=512, out_ch=512, kernel = 3x3, padding=1)
    MaxPool2d(kernel_size = 2x2)
    Conv2d(in_ch=512, out_ch=512, kernel = 3x3, padding=1)
    Conv2d(in_ch=512, out_ch=512, kernel = 3x3, padding=1)
    MaxPool2d(kernel_size = 2x2)
    return Feature Map 8x16x512
End function
```

Selanjutnya, dilakukan pembuatan untuk *decoder* dari UNet-ConvLSTM. Berikut adalah *pseudocode* yang digunakan untuk membuat model *decoder* untuk UNet-ConvLSTM.

ALGORITMA 10: Function Decoder UNet

```
Input: Feature map with size of 8x6x512
Initialization: Randomly initiate weight and bias of convolution filters
Output: 2-channels image: Predicted Background and Lane
Function Decoder (Input):
    ConvTranspose2d(in_ch = 1024, out_ch=256)
    Layer 1 = Concatenate Feature from Corresponding Encoder Layer
    ConvTranspose2d(in_ch = 512, out_ch=128)
    Layer 2 = Concatenate Feature from Corresponding Encoder Layer
    ConvTranspose2d(in_ch = 256, out_ch=64)
    Layer 3 = Concatenate Feature from Corresponding Encder Layer
    Layer 4 = ConvTranspose2d(in_ch = 256, out_ch=64)
    Conv2d(in_ch=64, out_ch=2)
```

```
10      Output 2-channels image
11  End
12  End function
```

Dengan mengacu pada detail susunan arsitektur pada subbab 2.2.6.2, dan menggunakan inisiasi yang sama untuk blok ConvLSTM dari model, berikut adalah kode untuk membuat keseluruhan arsitektur dari UNet-ConvLSTM.

ALGORITMA 11: Arsitektur UNet-ConvLSTM

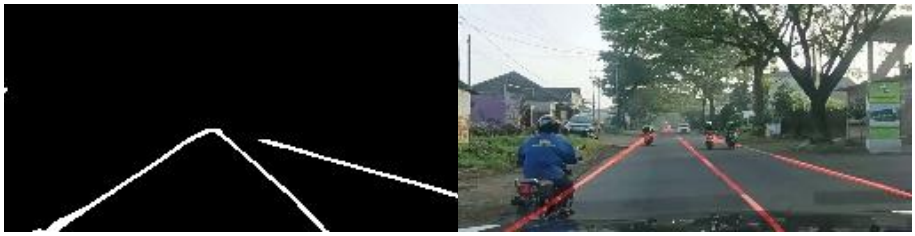
```
    Input: Frame from driving scene video with size of 128x256
    Initialization: Randomly initiate weight and bias of convolution filters
    Output: 2-channels image: Predicted Background and Lane
1   listOfFrame = [ ]
2   for frame in FiveConsecutiveFrames:
3       Enc = function Encoder UNet (frame)
4       listOfFrame.append(Enc)
5   ConvLSTM = function ConvLSTM (listOfFrame)
6   Dec = function Decoder UNet (ConvLSTM)
7   Output 2-channels image
8   End
```

Setelah kedua model terinisiasi, dilakukan proses *training* dan *validation* pada kedua arsitektur *neural network* ini dengan menggunakan dataset lajur sebagaimana tertera pada subbab 3.1.2. *Pseudocode* algoritma untuk proses *training* ditunjukkan pada algoritma 11.

ALGORITMA 12: Training Model Hybrid CNN-RNN

```
    Input: Minibatches from Dataset
    Initialization: Batch size, Neural Network Model, Optimizer, Learning Rate, Loss Function
                    Randomly Initiate Weight and Biases, Epochs
    Output: Trained Neural Network Model
1   for epoch in Epochs:
2       for batch in BatchedDataset:
3           output = Model(batch)
4           calculateLoss(output, ground truth)
5           Update Weight and Biases using Optimizer and LearningRate
6   End
```

Dalam prosesnya, setelah citra masukan melewati *encoder*, ConvLSTM, dan *decoder*, *feature map* yang dihasilkan akan melewati fungsi aktivasi softmax yang memiliki 2 kelas: *background* dan lajur. Hasil deteksi lajur ini kemudian akan digabungkan dengan citra masukan sehingga akan menghasilkan keluaran berupa citra masukan yang lajurnya telah terdeteksi. Proses ini dilakukan dengan cara mengubah warna map hasil deteksi menjadi warna merah. Hasil deteksi dan *overlay* antara gambar asli dan hasil deteksi ditunjukkan pada gambar 3.2.



Gambar 3.2 Hasil Deteksi dan *Overlay Image*

3.1.4 Desain Eksperimen dan Simulasi

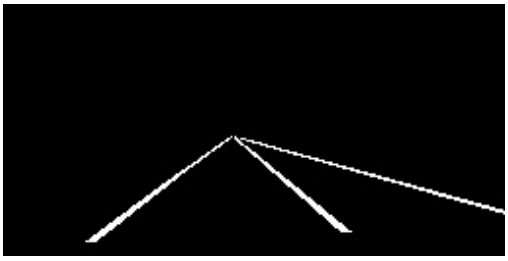
Tahap selanjutnya setelah dilakukan pembuatan model *hybrid* CNN-RNN adalah proses *training* dan *validation* dari model tersebut. Pada tahap ini, dilakukan beberapa inisialisasi parameter *training* seperti *epoch, bach size, optimizer, learning rate*, dan *early stop patience*. Proses *validation* dilakukan bersamaan ketika satu *epoch* pada *training* selesai dilakukan. Performa yang dihasilkan pada proses *validation* ini akan menjadi penentu dari *early stop patience*.

Setelah dilakukan proses *training* dan *validation*, model *hybrid* CNN-RNN ini akan diuji/*testing* dengan menggunakan dataset di luar dataset yang digunakan dalam proses *training* dan *validation*. Dataset *testing* yang digunakan diambil secara mandiri melalui kamera *smartphone* dengan resolusi 720p dan *frame rate* 30fps dan terdiri dari sembilan kombinasi kondisi jalan dan cuaca:

1. Jalan lurus, kondisi cuaca cerah
2. Jalan lurus, kondisi cuaca hujan
3. Jalan lurus, kondisi malam hari
4. Belok kanan, kondisi cuaca cerah
5. Belok kanan, kondisi cuaca hujan
6. Belok kanan, kondisi malam hari
7. Belok kiri, kondisi cuaca cerah
8. Belok kiri, kondisi cuaca hujan
9. Belok kiri, kondisi malam hari

Data yang diambil berupa video tersebut kemudian akan diekstrak tiap *frame*-nya dan dilakukan proses *labelling* sebagai *ground truth* untuk evaluasi. Proses *labelling* dilakukan dengan menggunakan bantuan *software* labelme. Gambar 3.3 dan 3.4 menunjukkan suatu *frame* yang sedang diberi label beserta *ground truth* hasil *labelling*.

Gambar 3.3 Proses *Labelling* Dataset Untuk *Testing*



Gambar 3.4 *Ground Truth* Hasil *Labelling*

3.1.5 Pengujian Algoritma dan Evaluasi

Pengujian algoritma pada penelitian ini dilakukan dengan cara mengevaluasi nilai F1-*score* dari masing-masing arsitektur. Hasil deteksi oleh SegNet-ConvLSTM atau UNet-ConvLSTM akan dibandingkan dengan label dari dataset untuk dievaluasi performanya. F1-*score* dipilih sebagai metrik performa utama karena proporsi lajur yang *imbalance* terhadap keseluruhan gambar. Akurasi tidak menjadi metrik utama dalam permasalahan ini karena akurasi akan tetap menghasilkan nilai yang tinggi meskipun model tidak mendeteksi lajur sama sekali pada citra masukan.

Setelah itu, diperlukan cara tertentu agar performa SegNet-ConvLSTM dan UNet-ConvLSTM dalam mendeteksi lajur dapat dibandingkan dengan *hough transform* (Widianto, 2020). *Mask* hasil deteksi lajur oleh *hough transform* harus diekstrak terlebih dahulu agar perhitungan F1-*score* berdasarkan label yang telah dibuat dapat dilakukan. Setelah nilai F1-*score* didapatkan, dilakukan perbandingan performa dari ketiga algoritma tersebut. Selain perhitungan F1-*score* dilakukan pula perhitungan *running time* dari masing-masing algoritma yang digunakan.

3.1.6 Penyusunan Laporan

Penyusunan laporan tugas akhir ini dilakukan setelah data-data yang relevan beserta kesimpulannya didapat. Penyusunan ini bertujuan untuk menyampaikan hasil penelitian yang telah dilakukan.

3.2 Peralatan yang Digunakan

Pembuatan model dilakukan dengan menggunakan bahasa pemrograman Python dengan bantuan *framework* PyTorch. PyTorch merupakan *library open source* yang dikembangkan oleh Facebook's AI Research Lab yang digunakan khusus untuk pengembangan *deep learning*.

PyTorch bekerja dengan dua komponen utama, yaitu tensor dan *computational graph*. Tipe data yang digunakan PyTorch adalah tensor yang mirip dengan array multidimensi. Tensor ini digunakan untuk menyimpan serta memanipulasi input, output, dan parameter model. *Neural network* pada PyTorch direpresentasikan sebagai *computational graph* yang mampu berjalan pada GPU sehingga komputasi dapat berjalan lebih cepat. Keunggulan PyTorch dibandingkan dengan *framework* yang lain adalah penggunaannya yang fleksibel serta sangat *Pythonic* sehingga sangat mempermudah developer yang sudah terbiasa dengan menggunakan bahasa Python.

Selain PyTorch, terdapat beberapa *library* lain yang menunjang dalam tugas akhir ini. Tabel 3.1 menunjukkan beberapa daftar *library* utama yang digunakan dalam tugas akhir ini.

Tabel 3.1 Bahasa Pemrograman dan *Library* yang Digunakan

Library	**Versi**
Python	3.7.11
PyTorch	1.10.2
OpenCV	4.5.5.62
Numpy	1.21.5
Matplotlib	3.5.1
jcopdl	1.1.9

BAB 4 HASIL DAN PEMBAHASAN

4.1 Parameter *Training Neural Network*

Beberapa inisiasi parameter perlu dilakukan, sebelum melakukan *training* pada arsitektur *Hybrid* CNN-RNN, agar *training* dapat berjalan maksimal serta membuat *error* sekecil mungkin. Parameter yang digunakan untuk proses *training* dari UNet-ConvLSTM dan SegNet-ConvLSTM ditunjukkan pada tabel 4.1

Tabel 4.1 Parameter Proses *Training Neural Network*

Parameter	Nilai
Epoch	50
Batch Size	4
Optimizer	Adam
Learning Rate	0.0001
Early Stop Patience	10

Parameter *epoch* menunjukkan jumlah dimana suatu proses *learning* telah melewati dan mempelajari keseluruhan dataset. Pada umunya, semakin banyak *epoch* yang digunakan, semakin baik performa *neural network* pada data *training*. Namun, perlu juga diperhatikan agar *neural network* tidak *overfitting* pada data *training* yang digunakan. Ketika melakukan *training* pada suatu *neural network* yang besar, terdapat suatu tahap di dalam proses *training* dimana model akan berhenti untuk melakukan generalisasi dan justru mempelajari *statistical noise* pada dataset *training*. Inilah kondisi yang disebut dengan *overfitting*. *Overfitting* pada dataset *training* akan meningkatkan *error* generalisasi dan membuat model menjadi kurang berguna ketika membuat prediksi pada data yang baru.

Salah satu cara untuk mengatasi *overfitting* adalah dengan menggunakan *early stop*. *Early stop* merupakan salah satu cara dimana proses *training* dilakukan dengan jumlah *epoch* yang besar dan kemudian pada tiap *epoch*-nya akan dievaluasi performanya dengan menggunakan dataset validasi. Proses *training* akan berhenti, ketika performa model pada dataset validasi ini menurun, Nilai 10 pada *early stop patience* yang digunakan untuk tugas akhir ini memiliki arti bahwa ketika performa suatu model pada dataset validasi tidak meningkat sebanyak 10 *epoch*, maka proses *training* akan dihentikan. Hal ini dilakukan untuk menjamin bahwa model yang didapatkan memiliki performa generalisasi yang baik.

Batch size adalah banyaknya kumpulan data yang disebarkan dan diproses pada setiap iterasi proses *learning*. Nilai *batch size* yang digunakan pada proses *training* ini adalah 4. Pemilihan *batch size* yang lebih dari satu ini bertujuan untuk mengoptimalkan komputasi paralel yang dilakukan oleh GPU. Nilai *batch size* ini tidak bisa lebih dari 4 karena kapasitas memori yang tidak cukup untuk melakukan komputasi di atas nilai tersebut.

Learning rate merupakan parameter *training* yang memiliki fungsi untuk melakukan *update* pada *weight* di *neural network* saat proses *training* dilakukan, Parameter ini memiliki rentang nilai antara 0 dan 1. Proses *training* akan berjalan semakin cepat namun kurang akurat, jika nilai dari parameter ini semakin besar, sebaliknya, semakin kecil nilai dari *learning rate*, proses *training* menjadi semakin akurat namun menjadi semakin lambat.

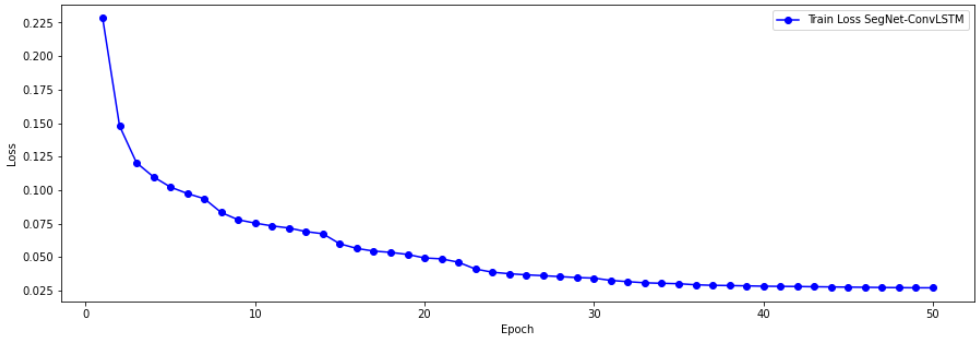
4.2 *Training Neural Network*

Proses pembuatan model dan *training Hybrid* CNN-RNN untuk tugas akhir ini dilakukan dengan menggunakan perangkat keras dengan spesifikasi yang tertera pada tabel 4.2.

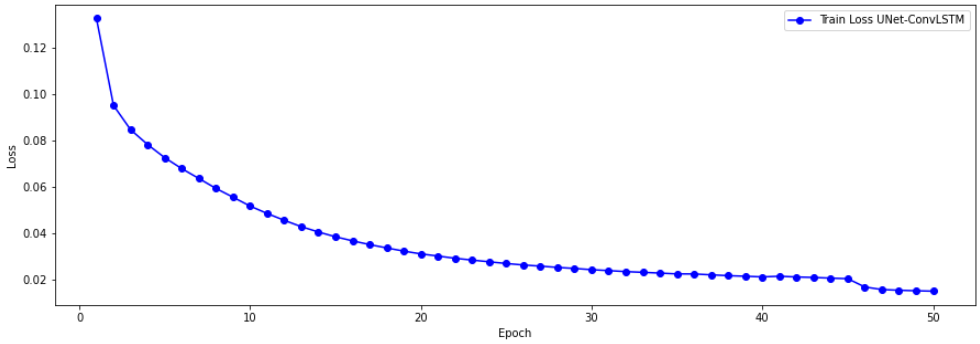
Tabel 4.2 Spesifikasi Perangkat Untuk *Training Neural Network*

Prosesor	AMD Ryzen 9 5900HS
RAM	16 GB
GPU	NVIDIA GeForce RTX 3060 Laptop
GPU Memory	6 GB
Memory Bandwidth	336 GB/s
NVIDIA Cuda Cores	3840

Terdapat dua jenis model *Hybrid* CNN-RNN yang di-*train* dalam tugas akhir ini: SegNet-ConvLSTM dan UNet-ConvLSTM. Kedua arsitektur ini menggunakan parameter *training* dan spesifikasi GPU yang sama serta dilakukan dengan menggunakan dataset tvtLane. Sebanyak 9548 gambar yang berlabel beserta augmentasinya digunakan dalam proses *training* ini. Grafik rata-rata *loss* pada proses *training* yang dilakukan pada kedua jenis arsitektur tersebut dapat dilihat pada gambar 4.1 dan 4.2.



Gambar 4.1 Grafik *Loss* pada Proses Training SegNet-ConvLSTM



Gambar 4.2 Grafik *Loss* Pada Proses *Training* UNet-ConvLSTM

Gambar 4.1 dan 4.2 dapat menjelaskan bahwa *loss* terbesar untuk setiap arsitektur yang digunakan terjadi pada *epoch* pertama, yaitu sebesar 0.2284 untuk SegNet-ConvLSTM dan 0.1327 untuk UNet-ConvLSTM. Nilai *loss* yang paling besar pada saat *epoch* pertama ini disebabkan karena model baru saja melakukan *training* dengan menggunakan dataset masukan.

Nilai *loss* untuk kedua arsitektur terus mengalami penurunan, seiring dengan bertambahnya jumlah *epoch*

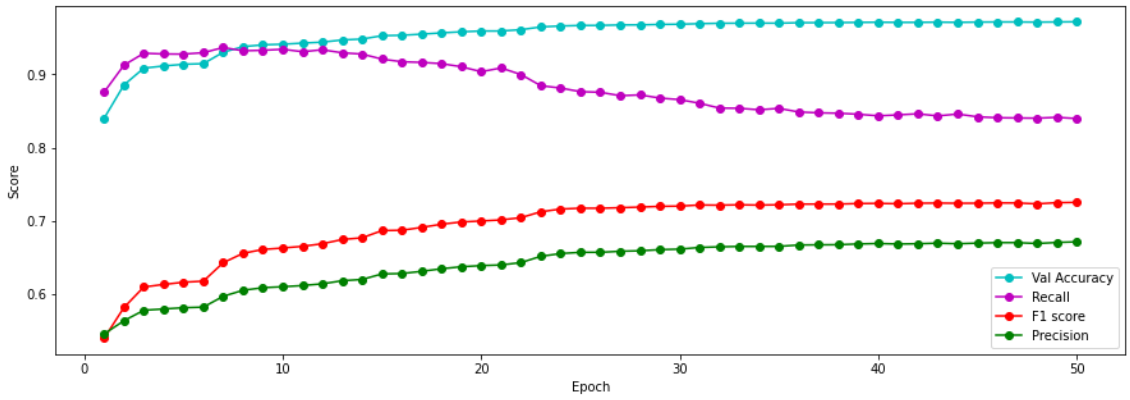
Setelah melewati *epoch* ke-40, penurunan *loss* menjadi sangat kecil bahkan sempat terjadi sedikit kenaikan nilai *loss* dari *epoch* ke-40 menuju ke-41 pada model UNet-ConvLSTM. Nilai *loss* sebesar 0.0210 pada *epoch* ke-40 meningkat sedikit menjadi 0.0213 pada *epoch* ke-41 sebelum akhirnya turun kembali pada *epoch* berikutnya. Karena penurunan *loss* yang sangat kecil dan nilainya mendekati nol, proses *training* dicukupkan hingga *epoch* ke-50. Selain melakukan *training* pada model, perlu dilakukan *validation* untuk melihat performa model dalam melakukan deteksi pada bagian dataset yang tidak digunakan untuk proses *training*. Proses *validation* ini bertujuan untuk memastikan bahwa model yang digunakan untuk proses *testing* nantinya adalah model yang memiliki performa yang paling baik.

4.3 *Validation Neural Network*

Model *hybrid* CNN-RNN yang telah di-*training* kemudian akan diuji performanya dengan menggunakan bagian dataset yang tidak digunakan saat proses *training*. Gambar dari dataset yang digunakan untuk proses *validation* ini sebanyak 1268 gambar. Terdapat beberapa parameter yang akan digunakan untuk melihat dan mengevaluasi performa model hasil *training*: *Accuracy*, *Recall*, *Precision*, dan F1-*score*. Keempat parameter tersebut memiliki rentang nilai dari 0 sampai 1. Nilai 0 berarti bahwa performa yang dihasilkan model adalah buruk. Sementara nilai 1 berarti bahwa performa yang dihasilkan model sangat akurat. Dari keempat parameter di atas, F1-*score* merupakan parameter utama yang menjadi acuan. Hal ini terjadi karena permasalahan deteksi lajur merupakan permasalahan klasifikasi biner yang *imbalance*. Jumlah piksel yang tergolong lajur pada suatu gambar masukan jauh lebih sedikit dibandingkan dengan piksel yang tergolong *background.* Dalam permasalahan deteksi lajur ini, jumlah *false positive* dan *false negative* diinginkan sekecil mungkin sehingga F1-*score* merupakan metrik utama yang dijadikan sebagai acuan dalam mengevaluasi model *hybrid* CNN-RNN. Epoch yang menghasilkan model dengan nilai F1-*score* terbaik kemudian akan dipilih untuk proses *testing* pada tahap berikutnya.

4.3.1 *Validation* SegNet-ConvLSTM

Hasil *validation* dari model SegNet-ConvLSTM untuk tiap *epoch*-nya ditunjukkan pada gambar 4.3.

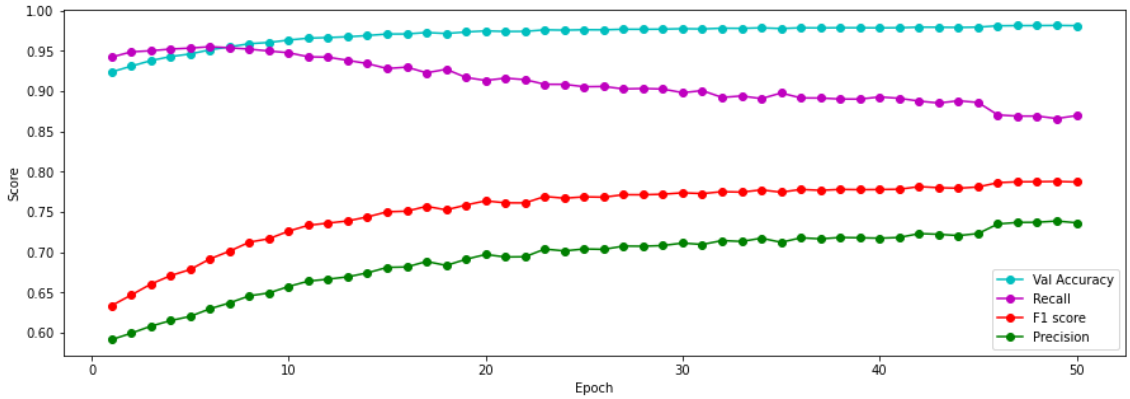


Gambar 4.3 Grafik *Validation* SegNet-ConvLSTM

Grafik *validation* SegNet-ConvLSTM pada gambar 4.3 menunjukkan bahwa *accuracy* terbesar terdapat pada *epoch* 50 dengan nilai sebesar 0.9718. Nilai *recall* terbesar terdapat pada *epoch* ke 7 dengan nilai sebesar 0.9367. Nilai *precision* terbesar dicapai pada *epoch* 50 dengan nilai 0.6711. Nilai F1-*score* terbesar dicapai pada saat *epoch* 50 dengan nilai 0.7251. Pada grafik tersebut, terlihat bahwa nilai *recall* cenderung turun seiring dengan bertambahnya jumlah *epoch*. Sebaliknya, terlihat juga bahwa nilai *precision* dan F1-*score* semakin meningkat seiring dengan bertambahnya jumlah *epoch*. Hal ini terjadi karena pada setiap *epoch*, model *neural network* dalam proses pembelajarannya berusaha untuk meningkatkan F1-*score* dengan cara meningkatkan nilai *precision*. Peningkatan nilai *precision* ini menyebabkan nilai *recall* secara otomatis menjadi menurun seiring dengan bertambahnya jumlah *epoch*. Model yang dihasilkan oleh *epoch* dengan F1-*score* tertinggi ini kemudian akan digunakan untuk proses *testing* dengan data yang berada di luar dataset yang digunakan untuk proses *training* maupun *validation*.

4.3.2 *Validation* UNet-ConvLSTM

Hasil *validation* dari model UNet-ConvLSTM untuk tiap *epoch*-nya ditunjukkan pada gambar 4.4.



Gambar 4.4 Grafik *Validation* UNet-ConvLSTM

Grafik *validation* UNet-ConvLSTM pada gambar 4.4 menunjukkan bahwa nilai *accuracy* terbesar terletak pada *epoch* 49 dengan nilai 0.9817. Nilai *recall* terbesar dicapai pada *epoch* 6 dengan nilai sebesar 0.955. Kemudian, nilai *precision* dan F1-*score* terbesar didapatkan pada *epoch* 49 dengan nilai masing-masing secara berurutan sebesar 0.7387 dan 0.7878. Sebagaimana pada proses *validation* SegNet-ConvLSTM, nilai *recall* dari UNet-ConvLSTM cenderung turun seiring dengan bertambahnya jumlah *epoch* dan nilai *precision* serta F1-*score* meningkat dengan bertambahnya jumlah *epoch*. Hal ini terjadi karena pada setiap *epoch*, model UNet-ConvLSTM dalam proses pembelajarannya berusaha untuk meningkatkan F1-*score* dengan cara meningkatkan nilai *precision*. Peningkatan nilai *precision* ini menyebabkan nilai *recall* secara otomatis menjadi menurun seiring dengan bertambahnya jumlah *epoch*.

Berdasarkan data yang diperoleh dari proses *validation* ini, performa UNet-ConvLSTM memiliki nilai yang lebih tinggi dibandingkan dengan SegNet-ConvLSTM yaitu dengan F1-*score* sebesar 0.7878 dibanding 0.7251. Hal ini dapat dikaitkan dengan penggunaan *skip-connections* pada arsitektur UNet-ConvLSTM sehingga dapat memperkaya informasi citra saat proses *upsampling*. Meskipun demikian, kedua model *hybrid* CNN-RNN ini tetap

dibandingkan performanya dalam proses *testing*, yaitu menguji performa model dengan data yang berasal dari luar dataset yang digunakan dalam proses *training* dan *validation*.

4.4 Pengambilan Data untuk *Testing Neural Network*

Data yang digunakan untuk *testing* model *neural network* merupakan data yang diambil secara mandiri dan terpisah dari dataset yang digunakan untuk *training* dan *validation*. Data yang digunakan dalam tugas akhir ini adalah video *driving scene* dengan sembilan kondisi jalan sebagaimana yang tertera pada subbab 3.3. Pengambilan video dilakukan di salah satu ruas jalan raya di Kota Malang, Jawa Timur. Kecepatan mobil saat pengambilan video berkisar di antara 20 hingga 30 km/jam. Perangkat yang digunakan untuk mengambil video adalah kamera *smartphone* dengan resolusi 720p dan *frame rate* 30fps. Kemudian, *frame-frame* dari video ini akan diekstraksi dan diberi label untuk bagian garis lajur pada jalan. Gambar 4.5 menunjukkan contoh *frame* hasil ekstraksi dari video dan labelnya.

Gambar 4.5 Contoh Gambar Untuk *Testing* Beserta Labelnya

4.5 *Testing* Model *Hybrid* CNN-RNN pada Beberapa Kondisi Jalan

Proses *testing* model *hybrid* CNN-RNN dilakukan pada kombinasi tiga jenis jalan: lurus, belok kanan, belok kiri, dan 3 jenis cuaca: cerah, hujan, dan malam hari. Performa dari SegNet-ConvLSTM dan UNet-ConvLSTM akan dilihat berdasarkan F1-*score* untuk masing-masing kombinasi kondisi jalan.

4.5.1 *Testing* pada Kondisi Cuaca Cerah dan Jalan Lurus

Model *hybrid* CNN-RNN pertama-tama diuji pada kondisi jalan lurus dengan kondisi cuaca yang cerah. Sebanyak 550 *frame* digunakan untuk proses *testing* ini. Pada kondisi jalan ini, SegNet-ConvLSTM menghasilkan *precision, recall,* dan F1-*score* berturut-turut sebesar 0.6457, 0.8454, 0.7010. Sementara itu, UNet-ConvLSTM menghasilkan *precision, recall,* dan F1-*score* berturut-turut sebesar 0.6934, 0.7900, dan 0.7387. Hal ini menunjukkan bahwa UNet-ConvLSTM memiliki performa yang lebih baik untuk kondisi jalan lurus dan cuaca yang cerah. Gambar 4.6 menunjukkan hasil deteksi dari SegNet-ConvLSTM dan UNet-ConvLSTM untuk kondisi jalan terkait.

Gambar 4.6 Hasil Deteksi Lajur oleh SegNet-ConvLSTM (kiri) dan UNet-ConvLSTM (kanan) pada Kondisi Jalan Lurus dan Cuaca Cerah.

Terdapat kondisi yang menghasilkan *false negative* maupun *false positive*, pada kondisi jalan lurus dan cuaca yang cerah. Sebagai contoh, deteksi lajur menjadi terganggu ketika berpapasan dengan kendaraan yang melintas dari arah yang berlawanan. Gambar 4.7 menunjukkan salah satu contoh kejadian tersebut.

Gambar 4.7 Hasil Deteksi Lajur oleh SegNet-ConvLSTM (kiri) dan UNet-ConvLSTM (kanan) yang Terganggu oleh Adanya Motor yang Sedang Lewat.

4.5.2 *Testing* pada Kondisi Cuaca Cerah dan Jalan Belok Kanan

Pengujian performa deteksi lajur pada kondisi cuaca cerah dan jalan belok kanan menggunakan sejumlah 334 *frame* yang diekstrak dari video. Pada kondisi ini, SegNet-ConvLSTM menghasilkan nilai *precision, recall,* dan F1-*score* sebesar 0.6502, 0.8454, dan 0.7060. Sementara itu, UNet-ConvLSTM menghasilkan nilai *precision, recall,* dan F1-*score* sebesar 0.6804, 0.7542, dan 0.7145. Hal ini menunjukkan bahwa UNet-ConvLSTM memiliki performa yang sedikit lebih baik untuk kondisi cuaca cerah dan jalan belok kanan. Gambar 4.7 menunjukkan hasil deteksi dari SegNet-ConvLSTM dan UNet-ConvLSTM untuk kondisi jalan terkait.

Gambar 4.8 Hasil Deteksi Lajur oleh SegNet-ConvLSTM (kiri) dan UNet-ConvLSTM (kanan) pada Kondisi Cuaca Cerah dan Jalan Belok Kanan.

Pada kondisi cuaca ini, masih terdapat beberapa kejadian *false positive* maupun *false negative*. Sebagai contoh, pada gambar 4.8, seharusnya terdapat tiga garis lajur. Namun, lajur bagian kanan tidak terdeteksi karena terdapat motor yang melintas sehingga menutupi garis lajur.

4.5.3 *Testing* pada Kondisi Cuaca Cerah dan Jalan Belok Kiri

Pengujian performa deteksi lajur kemudian juga dilakukan pada kondisi jalan belok kiri dengan cuaca cerah. Sebanyak 565 *frame* yang diekstrak dari video digunakan sebagai data *testing*. Pada kondisi ini, SegNet-ConvLSTM menghasilkan nilai *precision, recall,* dan F1-*score* sebesar 0.6318, 0.8291, dan 0.6840. Sementara itu, UNet-ConvLSTM menghasilkan nilai

precision, recall, dan F1-*score* sebesar 0.6630, 0.7288, dan 0.6924. Hal ini menunjukkan bahwa UNet-ConvLSTM memiliki performa yang sedikit lebih baik dibandingkan dengan SegNet-ConvLSTM. Gambar 4.9 menunjukkan hasil deteksi lajur untuk kondisi jalan ini. Terdapat beberapa kejadian *false positive* dan *false negative* yang terjadi saat proses deteksi lajur. Sebagai contoh, pada gambar 4.9, terdapat bagian yang seharusnya terdeteksi lajur tetapi tidak terdeteksi (garis lajur bagian kiri). Hal ini dimungkinkan terjadi karena lajur sedikit terputus dengan bagian depannya sehingga proses deteksi kurang maksimal. Selain itu adanya kendaraan yang melintas berlawanan arah juga menyebabkan bagian yang bukan lajur terdeteksi sebagai lajur.

Gambar 4.9 Hasil deteksi lajur oleh SegNet-ConvLSTM (kiri) dan UNet-ConvLSTM (kanan) pada kondisi cuaca cerah dan jalan belok kiri.

4.5.4 *Testing* pada Kondisi Cuaca Hujan dan Jalan Lurus

Pengujian performa deteksi lajur juga dilakukan pada kondisi cuaca yang sedang hujan. Pengujian performa deteksi lajur pada kondisi cuaca hujan dan jalan lurus menggunakan sebanyak 550 *frame* yang diekstrak dari video yang diambil. Pada kondisi ini, SegNet-ConvLSTM menghasilkan nilai *precision, recall,* dan F1-*score* sebesar 0.6448, 0.7383, 0.6705 sedangkan UNet-ConvLSTM menghasilkan *precision, recall,* dan F1-*score* sebesar 0.6691, 0.6567, dan 0.6609. Nilai tersebut menunjukkan bahwa SegNet-ConvLSTM memiliki performa yang lebih baik.

Gambar 4.10 Hasil Deteksi Lajur oleh SegNet-ConvLSTM (kiri) dan UNet-ConvLSTM (kanan) pada Kondisi Cuaca Hujan dan Jalan Lurus.

Nilai F1-*score* yang dihasilkan oleh kedua algoritma pada pengujian performa deteksi lajur dalam kondisi cuaca yang sedang hujan cenderung menurun jika dibandingkan dengan cuaca yang cerah. Salah satu faktor yang menyebabkan penurunan nilai F1-*score* ini adalah adanya genangan air hujan yang menimbulkan pantulan pada jalan. Hal ini menyebabkan garis lajur pada jalan tidak terlihat secara jelas sehingg deteksi yang dilakukan menjadi tidak maksimal.

Sebagai contoh, pada gambar 4.10, garis lajur pada bagian kanan jalan tidak terdeteksi secara maksimal saat adanya genangan air yang cukup banyak.

4.5.5 *Testing* pada Kondisi Cuaca Hujan dan Jalan Belok Kanan

Pengujian performa deteksi lajur selanjutnya dilakukan pada kondisi cuaca hujan dan jalan belok kanan. Sebanyak 330 *frame* digunakan sebagai data pengujian. Pada kondisi ini, SegNet-ConvLSTM menghasilkan nilai *precision, recall,* dan F1-*score* sebesar 0.6496, 0.7257, dan 0.6715. Sementara itu, UNet-ConvLSTM menghasilkan nilai *precision, recall,* dan F1-*score* sebesar 0.6818, 0.6550, dan 0.6673. Nilai ini menunjukkan bahwa SegNet-ConvLSTM memiliki performa yang sedikit lebih baik dibandingkan UNet-ConvLSTM. Contoh hasil deteksi pada kondisi ini dapat dilihat pada gambar 4.11. Dari gambar tersebut terlihat bahwa ketiga garis lajur pada jalan tidak dapat terdeteksi secara maksimal. Hal ini terjadi karena banyaknya genangan air yang terdapat pada jalan terutama pada saat tikungan. Genangan ini menyebabkan garis lajur tidak terlihat begitu jelas sehingga performa deteksi menurun jika dibandingkan saat kondisi cuaca cerah.

Gambar 4.11 Hasil Deteksi Lajur oleh SegNet-ConvLSTM (kiri) dan UNet-ConvLSTM (kanan) pada Kondisi Cuaca Hujan dan Jalan Belok Kanan.

4.5.6 *Testing* pada Kondisi Cuaca Hujan dan Jalan Belok Kiri

Proses *testing* berikutnya adalah deteksi lajur pada kondisi cuaca hujan dan jalan belok kiri. Jumlah *frame* yang digunakan untuk *testing* pada kondisi ini adalah 569 *frame*. Pada kondisi ini, SegNet-ConvLSTM menghasilkan *precision, recall,* dan F1-*score* sebesar 0.6275, 0.7398, dan 0.6607, sedangkan UNet-ConvLSTM menghasilkan *precision, recall,* dan F1-*score* sebesar 0.6525, 0.6425, dan 0.6480. Dari nilai F1-*score* ini, dapat disimpulkan bahwa SegNet-ConvLSTM memiliki performa yang lebih baik dibandingkan UNet-ConvLSTM pada keseluruhan kondisi cuaca hujan. Meski demikian, masih terdapat kejadian dimana lajur tidak dapat terdeteksi dengan baik seperti pada gambar 4.12. Pada gambar 4.12 tersebut terlihat bahwa garis lajur tidak sepenuhnya terdeteksi oleh model *hybrid* CNN-RNN. Hal ini disebabkan karena genangan air yang cukup signifikan pada jalan.

Gambar 4.12 Hasil Deteksi Lajur oleh SegNet-ConvLSTM (kiri) dan UNet-ConvLSTM (kanan) pada Kondisi Cuaca Hujan dan Jalan Belok Kiri.

4.5.7 *Testing* pada Kondisi Malam Hari dan Jalan Lurus

Setelah melakukan *testing* untuk kondisi cerah dan hujan, model *hybrid* CNN-RNN juga diuji untuk kondisi malam untuk berbagai kondisi jalan. Pada kondisi malam hari dan jalan lurus, sebanyak 550 *frame* digunakan untuk proses *testing* ini. Gambar 4.13 menunjukkan contoh hasil deteksi pada kondisi ini.

Gambar 4.13 Hasil Deteksi Lajur oleh SegNet-ConvLSTM (kiri) dan UNet-ConvLSTM (kanan) pada Kondisi Malam Hari dan Jalan Lurus.

Deteksi lajur oleh SegNet-ConvLSTM (kiri) dan UNet-ConvLSTM (kanan) pada kondisi malam hari dan jalan lurus menunjukkan bahwa SegNet-ConvLSTM menghasilkan nilai *precision, recall,* dan F1-*score* sebesar 0.6577, 0.7330, 0.6791 sedangkan UNet-ConvLSTM menghasilkan nilai *precision, recall,* dan F1-*score* sebesar 0.6981, 0.6944, dan 0.6885. Nilai ini menunjukkan bahwa UNet-ConvLSTM memiliki performa yang sedikit lebih baik dibandingkan dengan SegNet-ConvLSTM. Gambar 4.13 memperlihatkan bahwa sebagian besar garis lajur dapat terdeteksi. Namun, terdapat bagian dari garis lajur yang paling kanan yang tidak terdeteksi. Penurunan kualitas deteksi ini bisa dikaitkan dengan intensitas pencahayaan yang hanya berasal dari mobil serta adanya kendaraan yang melintas dari arah berlawanan dengan kondisi sambil menyalakan lampunya. Kondisi ini menyebabkan citra dari garis lajur menjadi sedikit terdistorsi dan deteksi yang dihasilkan kurang maksimal.

4.5.8 *Testing* pada Kondisi Malam Hari dan Jalan Belok Kanan

Proses *testing* berikutnya adalah pengujian performa deteksi lajur pada kondisi malam hari dan jalan belok kanan. Banyak *frame* yang digunakan untuk *testing* pada kondisi ini adalah 330 *frame*. Pada kondisi ini, SegNet-ConvLSTM menghasilkan nilai *precision, recall,* dan F1-*score* sebesar 0.6785, 0.7239, dan 0.6918. Sedangkan UNet-ConvLSTM yang menghasilkan nilai *precision, recall,* dan F1-*score* sebesar 0.7118, 0.6171, dan 0.6570. Contoh hasil deteksi pada kondisi malam hari dan jalan belok kanan ditunjukkan pada gambar 4.14.

Gambar 4.14 Hasil Deteksi Lajur oleh SegNet-ConvLSTM (kiri) dan UNet-ConvLSTM (kanan) pada Kondisi Malam Hari dan Jalan Belok Kanan.

Gambar 4.14 memperlihatkan bahwa garis lajur tidak sepenuhnya terdeteksi. Hal ini dimungkinkan karena penerangan yang digunakan cukup sedikit yaitu hanya bersumber dari lampu sorot mobil saja. Penerangan yang sedikit ini menyebabkan citra dari garis lajur tidak dapat terlihat secara maksimal.

4.5.9 *Testing* pada Kondisi Malam Hari dan Jalan Belok Kiri

Proses testing yang terakhir adalah pengujian performa deteksi lajur pada kondisi malam hari dan jalan belok kiri. Banyak frame yang digunakan untuk testing pada kondisi ini adalah 655 frame. Pada kondisi ini, SegNet-ConvLSTM menghasilkan nilai *precision, recall,* dan F1-*score* sebesar 0.6085, 0.6252, dan 0.6037 sedangkan UNet-ConvLSTM menghasilkan nilai *precision, recall,* dan F1-*score* sebesar 0.6492, 0.5947, dan 0.6083. Nilai ini menunjukkan bahwa UNet-ConvLSTM memiliki performa yang sedikit lebih baik pada kondisi ini. Contoh hasil deteksi pada kondisi malam hari dan jalan belok kiri ditunjukkan pada gambar 4.15.

Pada gambar 4.15, terlihat bahwa garis lajur paling kanan tidak terdeteksi dengan baik. Hal ini disebabkan oleh adanya kendaraan yang melintas disertai dengan penggunaan lampu sorot dari arah yang berlawanan. Kondisi ini menyebabkan lajur tidak bisa terlihat dengan baik sehingga hasil deteksi yang dihasilkan oleh SegNet-ConvLSTM maupun UNet-ConvLSTM tidak maksimal

Gambar 4.15 Hasil Deteksi Lajur oleh SegNet-ConvLSTM (kiri) dan UNet-ConvLSTM (kanan) pada Kondisi Malam Hari dan Jalan Belok Kiri

4.6 Perbandingan Performa dengan *Hough Transform*

Setelah performa dari model *Hybrid* CNN-RNN untuk kesembilan kondisi didapatkan, dilakukan pembandingan performa deteksi lajur antara *Hybrid* CNN-RNN dengan algoritma *Hough Transform* (Widianto, 2020). Pada penelitian yang telah dilakukan oleh Widianto tersebut, evaluasi F1-*score* untuk hasil deteksi tidak dilakukan. Agar model *Hybrid* CNN-RNN dan *Hough Transform* dapat dikomparasikan, perhitungan F1-*score* untuk hasil deteksi dari *Hough Transform* perlu dilakukan. Perhitungan tersebut dapat dilakukan dengan cara mengekstrak segmen lajur dari hasil deteksi oleh *hough transform* dan kemudian membandingkannya dengan *ground truth* di masing-masing kondisi. Gambar 4.16 menunjukkan proses untuk mengambil segmen lajur dari hasil deteksi yang dilakukan oleh *hough transform*. Dengan adanya segmen lajur pada gambar 4.4 bagian kanan, perhitungan nilai F1-score dengan menggunakan persamaan 2.33 bisa dengan mudah dilakukan.

Gambar 4.16 Hasil Deteksi *Hough Transform* Beserta Hasil Ekstraksi Segmen Lajur

Tabel 4.3 menunjukkan perbandingan performa *hybrid* CNN-RNN dengan *hough transform* berdasarkan nilai F1-*score* masing-masing. Dari tabel 4.3 dapat terlihat bahwa baik UNet-ConvLSTM maupun SegNet-ConvLSTM memiliki nilai F1-*score* yang jauh di atas *hough transform*.

Tabel 4.3 Perbandingan F1-score *Hybrid* CNN-RNN dengan *Hough Transform*

Kondisi Jalan		**F1-Score**		
		UNet-ConvLSTM	**SegNet-ConvLSTM**	**Hough Transform**
Cerah	Lurus	0.7387	0.7010	0.1860
	Belok Kanan	0.7145	0.7060	0.1245
	Belok Kiri	0.6924	0.6840	0.2702
Hujan	Lurus	0.6609	0.6705	0.0630
	Belok Kanan	0.6673	0.6715	0.0472
	Belok Kiri	0.6480	0.6607	0.1410
Malam	Lurus	0.6885	0.6791	0.2059
	Belok Kanan	0.6570	0.6918	0.2019
	Belok Kiri	0.6083	0.6037	0.1508
Rata-rata		0.6751	0.6742	0.1545

Berdasarkan tabel 4.3, nilai F1-*score* yang tinggi dari UNet-ConvLSTM dan SegNet-ConvLSTM jika dibandingkan dengan *hough transform* ini dapat terjadi karena beberapa alasan. Pertama, *hough transform* hanya mendeteksi sebanyak dua garis lajur meskipun pada jalan seharusnya terdapat tiga garis lajur. Contoh kasus ini dapat terlihat pada gambar 4.17. Hal ini terjadi karena pada algoritma *hough transform*, dilakukan pendefinisian *region of interest* (ROI) secara *fixed*. Bentuk ROI yang digunakan dalam algoritma *hough transform* adalah segitiga. Adanya ROI yang bersifat *fixed* ini menyebabkan garis lajur yang berada di luar ROI tidak dapat terdeteksi oleh algoritma *hough transform* sehingga nilai F1-*score* yang dihasilkan menjadi kecil.

Gambar 4.17 Perbandingan Hasil Deteksi SegNet-ConvLSTM (kiri), UNet-ConvLSTM (tengah) dan *Hough Transform* (kanan) Untuk Kondisi Cuaca Cerah dan Jalan Lurus

Kedua, ketika melewati jalan yang memiliki tikungan, *hough transform* tidak menghasilkan hasil deteksi yang mengikuti kelengkungan jalan. Hasil deteksi yang tidak bisa mengikuti kelengkungan jalan ini juga disebabkan karena ROI yang digunakan tidak bersifat adaptif sehingga menyebabkan banyaknya *false negative* dan *false positive* yang terjadi. Tingginya nilai *false negative* dan *false positive* ini menyebabkan nilai F1-*score* yang dihasilkan oleh *hough transform* menjadi rendah. Gambar 4.18 menunjukkan contoh kasus ketika *hough transform* tidak bekerja baik pada jalan tikungan.

Gambar 4.18 Perbandingan Hasil Deteksi SegNet-ConvLSTM (Kiri), UNet-ConvLSTM (tengah) dan *Hough Transform* (kanan) untuk Kondisi Cuaca Cerah dan Belok Kanan.

Selain itu, berdasarkan tabel 4.3, terlihat juga bahwa UNet-ConvLSTM memiliki performa yang paling baik pada keseluruhan cuaca cerah, pada malam hari kondisi jalan lurus dan belok kiri. Sedangkan pada kondisi hujan dan malam hari belok kanan, SegNet-ConvLSTM memiliki performa yang lebih baik. Performa SegNet-ConvLSTM yang lebih baik pada kondisi hujan dan malam hari belok kanan ini bisa jadi disebabkan oleh kemampuan generalisasi arsitektur ini dalam kondisi tersebut. Sebagai contoh, pada gambar 4.12, SegNet-ConvLSTM tetap menghasilkan deteksi walaupun lebar garis hasil deteksinya lebih lebar daripada garis lajur sebenarnya. Sementara pada UNet-ConvLSTM deteksi lajur dilakukan dengan presisi yang semakismal mungkin sehingga lebar garis deteksi yang dihasilkan hampir sama dengan lebar garis sebenarnya pada gambar masukan, namun menjadi sangat mudah terganggu dengan adanya air hujan.

Performa SegNet-ConvLSTM yang lebih baik di kondisi hujan dan malam belok kanan ini juga bisa dikaitkan dengan nilai *precision* dan *recall* pada masing-masing arsitektur untuk kondisi tersebut. Terlihat bahwa meskipun nilai *precision* UNet-ConvLSTM sedikit lebih tinggi untuk tiap kondisi jalan pada saat hujan, nilai *recall* dari SegNet-ConvLSTM jauh lebih tinggi dibandingkan dengan UNet-ConvLSTM. Perbedaan yang cukup jauh pada nilai *recall* ini menyebabkan F1-*score* dari SegNet-ConvLSTM menjadi lebih tinggi. Hal ini sesuai dengan persamaan 2.32 yang menyatakan menunjukkan bahwa semakin tinggi nilai *recall*, semakin sedikit *false negative* yang terjadi pada deteksi lajur.

Pada tabel 4.3, dapat diketahui bahwa pada saat kondisi jalan lurus, performa kedua arsitektur justru lebih baik pada malam hari dibandingkan dengan pada saat hujan. Hal ini disebabkan karena pada saat hujan, genangan yang terjadi pada jalan lurus tersebut cukup banyak dan terdapat beberapa kendaraan yang melintas dari belakang maupun dari depan. Sementara pada kondisi malam hari dan jalan lurus, tidak terdapat genangan air, jumlah kendaraan yang melintas lebih sedikit, serta penerangan masih cukup terbantuk dengan lampu jalan yang ada. Gambar 4.19 menunjukkan perbedaan kondisi jalan antara jalan lurus pada kondisi hujan dan jalan lurus pada kondisi malam hari.

4.7 Pengaruh Penggunaan ConvLSTM Terhadap Performa Deteksi Lajur

Agar pengaruh penggabungan RNN ke dalam arsitektur *encoder-decoder* CNN dapat terlihat, dilakukan pembandingan performa deteksi lajur antara *Hybrid* CNN-RNN dan CNN dalam melakukan deteksi lajur. Arsitektur CNN yang digunakan untuk perbandingan adalah berupa SegNet dan UNet yang telah di-*pretrain* dengan dataset yang sama dan konfigurasi *training* yang sama. Tabel 4.4 menunjukkan perbandingan *Hybrid* CNN-RNN dan CNN untuk proses deteksi lajur.

Tabel 4.4 Perbandingan *Hybrid* CNN-RNN dengan CNN

Kondisi Jalan		**F1-Score**			
		UNet-ConvLSTM	**SegNet-ConvLSTM**	**UNet**	**SegNet**
Cerah	Lurus	0.7387	0.7010	0.7055	0.6903
	Belok Kanan	0.7145	0.7060	0.7049	0.6901
	Belok Kiri	0.6924	0.6840	0.6898	0.6816
Hujan	Lurus	0.6609	0.6705	0.6461	0.6175
	Belok Kanan	0.6673	0.6715	0.6329	0.6144
	Belok Kiri	0.6480	0.6607	0.6420	0.5909
Malam	Lurus	0.6885	0.6791	0.6383	0.6275
	Belok Kanan	0.6570	0.6918	0.5709	0.6108
	Belok Kiri	0.6083	0.6037	0.5480	0.5505
Rata-rata		0.6751	0.6742	0.6420	0.6304

Berdasarkan tabel 4.4, terlihat bahwa performa dari perbedaan performa antara *Hybrid* CNN-RNN dengan CNN tidak berbeda jauh pada saat kondisi cerah. Namun pada saat kondisi hujan dan malam hari, *Hybrid* CNN-RNN memiliki performa yang lebih baik untuk semua kondisi jalan. Nilai F1-*score* dari UNet-ConvLSTM pada saat kondisi malam hari dan jalan belok kanan, misalnya, memiliki nilai yang 15% lebih tinggi dibandingkan dengan hanya menggunakan UNet saja.

Nilai F1-*score* pada tabel 4.4 menunjukkan bahwa penggunaan RNN, dalam hal ini adalah ConvLSTM, mampu mengatasi penurunan performa yang cukup drastis dari CNN terutama pada kondisi hujan dan malam hari. Hal ini dimungkinkan karena pada struktur arsitekturnya, ConvLSTM menggunakan informasi-informasi berupa *cell state* dan *hidden state* dari *timestep* lampau untuk menghasilkan prediksi pada *timestep* tertentu sebagaimana tertera pada persamaan 2.24 sampai 2.29.

4.8 Pengujian Running Time Algoritma

Selain melakukan perbandingan untuk melihat nilai F1-*score* dari masing-masing algoritma, perlu dilakukan pengujian terkait *running time* dari masing-masing algoritma karena *running time* merupakan salah satu aspek penting ketika algoritma akan diimplementasikan secara langsung ke suatu mobil otonom. Lama *running time* ini bergantung pada perangkat keras yang digunakan.

Gambar 4.19 Perbedaan Kondisi Jalan yang Menjadi Salah Satu Faktor Performa Deteksi Lajur

Pada pengujian *running time* ini, perangkat yang digunakan memiliki spesifikasi sebagaimana yang tertera pada tabel 4.2. Hasil perbandingan *running time* ketiga algoritma dapat dilihat pada tabel 4.4.

Tabel 4.5 Perbandingan *Running Time Hybrid* CNN-RNN dengan *Hough Transform*

Algoritma	**Kondisi Jalan**	**Running Time (detik)**			**Rata-Rata Running Time Keseluruhan**
		Terlama	**Tercepat**	**Rata-rata**	
UNet-ConvLSTM	Cerah Lurus	0.2301	0.0250	0.0279	0.0281 detik
	Cerah Kanan	0.2300	0.0250	0.0292	
	Cerah Kiri	0.2291	0.0250	0.0280	
	Hujan Lurus	0.2301	0.0250	0.0279	
	Hujan Kanan	0.2311	0.0250	0.0294	
	Hujan Kiri	0.2321	0.0250	0.0279	
	Malam Lurus	0.2311	0.0250	0.0280	
	Malam Kanan	0.2301	0.0250	0.0296	
	Malam Kiri	0.2198	0.0210	0.0250	
SegNet-ConvLSTM	Cerah Lurus	0.2876	0.0290	0.0329	0.0327 detik
	Cerah Kanan	0.2881	0.0250	0.0296	
	Cerah Kiri	0.2861	0.0290	0.0329	
	Hujan Lurus	0.2871	0.0290	0.0346	
	Hujan Kanan	0.2866	0.0290	0.0345	
	Hujan Kiri	0.2871	0.0290	0.0328	
	Malam Lurus	0.2871	0.0290	0.0329	
	Malam Kanan	0.2871	0.0290	0.0346	
	Malam Kiri	0.2497	0.0260	0.0294	
UNet	Cerah Lurus	0.1966	0.0202	0.0228	0.0245 detik
	Cerah Kanan	0.1876	0.0203	0.0247	
	Cerah Kiri	0.2061	0.0209	0.0250	
	Hujan Lurus	0.1932	0.0220	0.0259	
	Hujan Kanan	0.2005	0.0217	0.0245	
	Hujan Kiri	0.2040	0.0191	0.0245	
	Malam Lurus	0.2098	0.0219	0.0257	
	Malam Kanan	0.2057	0.0193	0.0235	
	Malam Kiri	0.2008	0.0206	0.0237	

SegNet	Cerah Lurus	0.2562	0.0222	0.0301	0.0302 detik
	Cerah Kanan	0.2547	0.0238	0.0301	
	Cerah Kiri	0.2531	0.0241	0.0303	
	Hujan Lurus	0.2548	0.0225	0.0301	
	Hujan Kanan	0.2550	0.0230	0.0302	
	Hujan Kiri	0.2546	0.0240	0.0301	
	Malam Lurus	0.2561	0.0228	0.0302	
	Malam Kanan	0.2531	0.0230	0.0301	
	Malam Kiri	0.2549	0.0208	0.0302	
Hough Transform	Cerah Lurus	0.1400	0.0334	0.0395	0.0401 detik
	Cerah Kanan	0.0660	0.0361	0.0429	
	Cerah Kiri	0.0730	0.0324	0.0388	
	Hujan Lurus	0.0731	0.0370	0.0434	
	Hujan Kanan	0.0742	0.0384	0.0462	
	Hujan Kiri	0.0751	0.0357	0.0417	
	Malam Lurus	0.0633	0.0313	0.0375	
	Malam Kanan	0.0690	0.0314	0.0384	
	Malam Kiri	0.0579	0.0278	0.0322	

Berdasarkan tabel 4.5, UNet memiliki waktu pemrosesan yang paling cepat yaitu dengan rata-rata 0.0245 detik. *Hough transform* memiliki waktu pemrosesan paling lama dengan rata-rata 0.0401 detik. *Hough transform* menghasilkan waktu pemrosesan paling lama karena citra masukan harus melewati banyak tahapan sebelum menghasilkan deteksi lajur seperti *greyscale transformation, gaussian blur, color mask selection, canny edge detection,* penentuan ROI, dan *hough transformation*.

Jika dibandingkan dengan CNN, model *Hybrid* CNN-RNN memiliki waktu pemrosesan yang sedikit lebih lama. Hal ini disebabkan oleh adanya blok ConvLSTM yang terdapat pada *Hybrid* CNN-RNN sehingga membuat jumlah *neuron* yang terdapat pada model ini menjadi lebih banyak. Jumlah *neuron* yang lebih banyak pada model *hybrid* CNN-RNN ini menyebabkan waktu komputasi dari model menjadi lebih lama.

Kamera yang digunakan untuk pengujian pada tugas akhir ini memiliki *frame rate* sebesar 30 *frame per second* (fps). Hal ini berarti bahwa kamera membutuhkan sekitar 0.0333 detik untuk mengambil gambar tiap *frame*-nya. Secara rata-rata, model *hybrid* CNN-RNN telah mampu bekerja dengan menggunakan kamera 30 fps karena waktu deteksi rata-ratanya yang lebih rendah dari 0.0333 detik. Namun, pada kondisi tertentu seperti pada saat model *neural network* pertama kali mendapatkan data masukan, waktu deteksi yang dibutuhkan menjadi sedikit lebih lama, sehingga model mengalami keterlambatan dalam mengeluarkan hasil deteksinya yang menyebabkan adanya *error* dari proses deteksi yang dilakukan.

4.9 Pemodelan Noise Pada Saat Hujan dan Malam Hari

Terdapat beberapa faktor yang mempengaruhi keberhasilan proses deteksi lajur. Adanya gangguan-gangguan eksternal seperti pantulan yang disebabkan oleh air hujan dan intensitas cahaya dapat mempengaruhi performa deteksi lajur yang dihasilkan. Maka dari itu, *noise* yang dihasilkan oleh kedua faktor tersebut perlu dimodelkan. Pemodelan *noise* ini dilakukan agar

citra dapat di-*filter* terlebih dahulu sesuai dengan statistik *noise* yang dimiliki sebelum masuk ke model *neural network*.

Dengan mengasumsikan bahwa citra yang diambil pada kondisi cerah merupakan citra yang tidak memiliki *noise* dan citra yang diambil pada kondisi hujan dan malam hari adalah citra yang memiliki *noise*. Diasumsikan juga bahwa *noise* yang terjadi bersifat aditif, maka dapat dimodelkan:

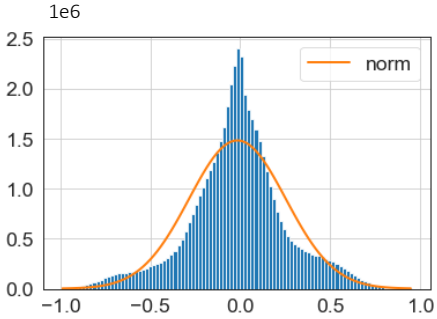
$$A(x,y) = B(x,y) + N(x,y) \tag{4.1}$$

Dimana $A(x,y)$ merupakan citra yang memiliki *noise*, $B(x,y)$ merupakan citra yang tidak memiliki noise, dan $N(x,y)$ merupakan *noise* pada citra. Maka, untuk mendapatkan model *noise* pada citra, dapat dilakukan:

$$N(x,y) = A(x,y) - B(x,y) \tag{4.2}$$

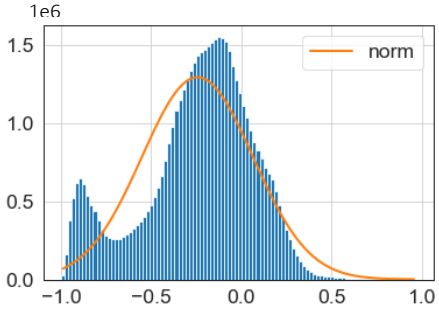
Berdasarkan persamaan 4.2, bisa didapatkan statistik *noise* seperti mean, standar deviasi, dan distribusi noise yang mendekati. Untuk mendapatkan model *noise* tersebut, masing-masing nilai piksel citra pada tiap kondisi dinormalisasi terlebih dahulu agar berada dalam rentang 0 sampai 1. Setelah itu, histogram dari selisih nilai piksel antara citra yang memiliki *noise* dan citra yang tidak memiliki noise dapat diplot dan distribusi yang paling mendekati data dapat dicari.

Gambar 4.20 adalah histogram dan distribusi *noise* yang paling mendekati untuk kondisi hujan. Didapatkan bahwa distribusi *noise* yang paling mendekati pada saat hujan adalah distribusi normal dengan $\mu = -0.0106$ dan $\sigma = 0.269$.



Gambar 4.20 Model *Noise* Pada Saat Hujan

Dengan cara yang sama, model *noise* pada saat kondisi malam hari juga dapat dicari. Gambar 4.21 menunjukkan model *noise* pada saat malam hari.



Gambar 4.21 Model Noise Pada Malam Hari

Pada gambar 4.21, terlihat pada histogram bahwa selisih piksel cenderung bernilai negatif serta terdapat banyak selisih piksel yang memiliki nilai yang mendekati -1. Hal ini terjadi karena pada saat malam hari, nilai piksel yang ditangkap kamera cenderung lebih rendah, bahkan mendekati nol (warna hitam) sehingga menyebabkan persamaan 4.2 menghasilkan nilai yang negatif. Jika dimodelkan dengan menggunakan distribusi normal, didapatkan $\mu = -0.2434$ dan $\sigma = 0.3080$. Kedua model *noise* baik pada saat hujan dan malam hari, menunjukkan nilai *mean* yang negatif. Hal ini berarti bahwa nilai piksel yang ditangkap pada kedua kondisi ini lebih rendah (lebih gelap) dibandingkan ketika pada saat kondisi cuaca cerah. Kondisi ini sesuai dengan intensitas cahaya yang memang lebih rendah ketika hujan dan malam hari.

Halaman ini sengaja dikosongkan

BAB 5 KESIMPULAN DAN SARAN

5.1 Kesimpulan

Berdasarkan hasil pengujian dan Analisa yang telah dilakukan, dapat diperoleh beberapa poin kesimpulan:

a. Model *hybrid* CNN-RNN berupa SegNet-ConvLSTM dan UNet-ConvLSTM mampu digunakan untuk deteksi lajur pada suatu jalan.
b. Performa dari algoritma untuk deteksi lajur dapat diukur dengan menggunakan F1-*score*. Pada saat *validation* maupun *testing*, UNet-ConvLSTM menghasilkan nilai F1-*score* rata-rata yaitu 0.7878 untuk *validation* dan 0.6751 untuk *testing*. Sementara itu, SegNet-ConvLSTM menghasilkan nilai F1-*score* sebesar 0.7251 untuk *validation* dan 0.6742 untuk *testing*.
c. Pada beberapa gambar masukan, masih terdapat sejumlah kesalahan seperti *false positive* maupun *false negative*. Kesalahan deteksi ini dapat disebabkan beberapa faktor, diantaranya adalah adanya genangan air yang signifikan pada jalan pada saat hujan dan minimnya penerangan saat malam hari.
d. Penggunaan ConvLSTM pada arsitektur mencegah penurunan performa yang signifikan pada kondisi hujan dan malam hari
e. Jika dibandingkan dengan *hough transform* dan CNN, F1-*score* dari UNet-ConvLSTM maupun SegNet-ConvLSTM memiliki nilai yang lebih tinggi. Nilai F1-*score* rata-rata untuk *hough transform* adalah 0.1545. Sementara CNN dalam bentuk UNet dan SegNet memiliki nilai F1-*score* sebesar 0.6420 dan 0.6304.
f. Waktu pemrosesan deteksi lajur paling cepat dicapai oleh UNet. Kemudian diikuti oleh UNet-ConvLSTM, SegNet, SegNet-ConvLSTM, dan Hough Transform secara berurutan. Secara garis besar, model *hybrid* CNN-RNN mampu digunakan pada kamera 30 fps dan dapat bekerja secara *real-time*.

5.2 Saran

Terdapat beberapa saran yang dapat diberikan untuk pengembangan lebih lanjut dari topik penelitian deteksi lajur:

a. Penambahan data untuk proses *training*. Dataset yang umumnya tersedia saat ini diambil di jalan yang tidak terlalu padat seperti jalan tol. Penambahan ini diharapkan dapat membuat performa dari model *neural network* yang digunakan menjadi lebih meningkat.
b. Pada pengembangan yang akan datang, diharapkan algoritma dapat dilatih untuk membedakan garis lajur yang putus-putus dengan yang tidak, mengingat kedua bentuk lajur tersebut memiliki arti yang berbeda
c. Arsitektur lain yang memiliki kemampuan generatif seperti *Generative Adversarial Networks* (GAN) dapat dikembangkan untuk merekayasa kondisi jalan yang tidak memiliki garis lajur sama sekali.
d. Penambahan objek deteksi dapat dilakukan seperti deteksi halangan maupun deteksi rambu-rambu pada jalan.
e. Dapat ditambahkan filter *wiener* pada citra sebelum masuk ke model *neural network* sesuai dengan model statistik *noise* yang telah didapatkan.

Halaman ini sengaja dikosongkan

DAFTAR PUSTAKA

Albawi, S., Mohammed, T. A., & Al-Zawi, S. (2018). Understanding of a convolutional neural network. *Proceedings of 2017 International Conference on Engineering and Technology, ICET 2017*, *2018-January*, 1–6. https://doi.org/10.1109/ICEngTechnol.2017.8308186

Alsallakh, B., Kokhlikyan, N., Miglani, V., Yuan, J., & Reblitz-Richardson, O. (2020). *Mind the Pad -- CNNs can Develop Blind Spots*. http://arxiv.org/abs/2010.02178

Badrinarayanan, V., Kendall, A., & Cipolla, R. (2015). SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *39*(12), 2481–2495. http://arxiv.org/abs/1511.00561

Dalianis, H. (2018). Clinical text mining: Secondary use of electronic patient records. In *Clinical Text Mining: Secondary Use of Electronic Patient Records*. Springer International Publishing. https://doi.org/10.1007/978-3-319-78503-5

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning*. MIT Press.

Hochreiter, S., & Schmidhuber, J. (1997). LONG SHORT-TERM MEMORY. *Neural Computation*, *9*(8), 1735–1780.

Huang, G., Liu, Z., van der Maaten, L., & Weinberger, K. Q. (2017). Densely Connected Convolutional Networks. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 4700–4708. https://github.com/liuzhuang13/DenseNet.

Jo, K., Kim, J., Kim, D., Jang, C., & Sunwoo, M. (2014). Development of autonomous car-part i: Distributed system architecture and development process. *IEEE Transactions on Industrial Electronics*, *61*(12), 7131–7140. https://doi.org/10.1109/TIE.2014.2321342

Mahmud Yusuf, M., Karim, T., & Saifuddin Saif, A. F. M. (2020, January 10). A robust method for lane detection under adverse weather and illumination conditions using convolutional neural network. *PervasiveHealth: Pervasive Computing Technologies for Healthcare*. https://doi.org/10.1145/3377049.3377105

Putra, I. W. S. E. (2016). *KLASIFIKASI CITRA MENGGUNAKAN CONVOLUTIONAL NEURAL NETWORK (CNN) PADA CALTECH 101* [Undergraduate Thesis]. Institut Teknologi Sepuluh Nopember.

Putra, J. W. G. (2020). *Pengenalan Konsep Pembelajaran Mesin dan Deep Learning* (1.4).

Ronneberger, O., Fischer, P., & Brox, T. (2015). U-net: Convolutional networks for biomedical image segmentation. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *9351*, 234–241. https://doi.org/10.1007/978-3-319-24574-4_28

SAE. (2018). *Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles*.

Shi, X., Chen, Z., Wang, H., Yeung, D.-Y., Wong, W.-K., & Woo, W.-C. (2015). Convolutional LSTM Network: A Machine Learning Approach for Precipitation Nowcasting. *Advances in Neural Information Processing Systems (NIPS)*, 802–810.

Taeihagh, A., & Lim, H. S. M. (2019). Governing autonomous vehicles: emerging responses for safety, liability, privacy, cybersecurity, and industry risks. *Transport Reviews*, *39*(1), 103–128. https://doi.org/10.1080/01441647.2018.1494640

Tang, J., Li, S., & Liu, P. (2021). A review of lane detection methods based on deep learning. *Pattern Recognition*, *111*. https://doi.org/10.1016/j.patcog.2020.107623

Widianto, S. C. (2020). *Deteksi Lajur Mobil Otonom Pada Kondisi Gambar Terdistorsi dan Kurang Pencahayaan Menggunakan Pengolahan Citra* [Undergraduate Thesis]. Institut Teknologi Sepuluh Nopember.

Zaremba, W., Sutskever, I., & Vinyals, O. (2014). *Recurrent Neural Network Regularization*. http://arxiv.org/abs/1409.2329

Zou, Q., Jiang, H., Dai, Q., Yue, Y., Chen, L., & Wang, Q. (2019). Robust Lane Detection from Continuous Driving Scenes Using Deep Neural Networks. *IEEE Transactions on Vehicular Technology*, *69*(1), 41–54. https://doi.org/10.1109/TVT.2019.2949603

LAMPIRAN

Program utils

```
import torch
import torch.nn as nn
import torch.nn.functional as F
import math
import numpy as np
from torch.autograd import Variable

class double_conv(nn.Module):
    '''(conv => BN => ReLU) * 2'''
    def __init__(self, in_ch, out_ch):
        super(double_conv, self).__init__()
        self.conv = nn.Sequential(
            nn.Conv2d(in_ch, out_ch, 3, padding=1),
            nn.BatchNorm2d(out_ch),
            nn.ReLU(inplace=True),
            nn.Conv2d(out_ch, out_ch, 3, padding=1),
            nn.BatchNorm2d(out_ch),
            nn.ReLU(inplace=True)
        )
    def forward(self, x):
        x = self.conv(x)
        return x
class inconv(nn.Module):
    def __init__(self, in_ch, out_ch):
        super(inconv, self).__init__()
        self.conv = double_conv(in_ch, out_ch)
    def forward(self, x):
        x = self.conv(x)
        return x
class down(nn.Module):
    def __init__(self, in_ch, out_ch):
        super(down, self).__init__()
        self.mpconv = nn.Sequential(
            nn.MaxPool2d(2),
            double_conv(in_ch, out_ch)
        )
    def forward(self, x):
        x = self.mpconv(x)
        return x
class up(nn.Module):
    def __init__(self, in_ch, out_ch, bilinear=True):
        super(up, self).__init__()
        #  would be a nice idea if the upsampling could be learned too,
        #  but my machine do not have enough memory to handle all those weights
        if bilinear:
            self.up = nn.UpsamplingBilinear2d(scale_factor=2)
        else:
            self.up = nn.ConvTranspose2d(in_ch//2, in_ch//2, 2, stride=2)
        self.conv = double_conv(in_ch, out_ch)
    def forward(self, x1, x2):
        x1 = self.up(x1)
        diffX = x1.size()[2] - x2.size()[2]
        diffY = x1.size()[3] - x2.size()[3]
        x2 = F.pad(x2, (diffX // 2, int(diffX / 2),
                        diffY // 2, int(diffY / 2)))
        x = torch.cat([x2, x1], dim=1)
        x = self.conv(x)
        return x
class outconv(nn.Module):
    def __init__(self, in_ch, out_ch):
        super(outconv, self).__init__()
        self.conv = nn.Conv2d(in_ch, out_ch, 1)
    def forward(self, x):
        x = self.conv(x)
        return x
```

```
class ConvLSTMCell(nn.Module):
    def __init__(self, input_size, input_dim, hidden_dim, kernel_size, bias):
        """
        Initialize ConvLSTM cell.

        Parameters
        ----------
        input_size: (int, int)
            Height and width of input tensor as (height, width).
        input_dim: int
            Number of channels of input tensor.
        hidden_dim: int
            Number of channels of hidden state.
        kernel_size: (int, int)
            Size of the convolutional kernel.
        bias: bool
            Whether or not to add the bias.
        """
        super(ConvLSTMCell, self).__init__()
        self.height, self.width = input_size
        self.input_dim = input_dim
        self.hidden_dim = hidden_dim
        self.kernel_size = kernel_size
        self.padding = kernel_size[0] // 2, kernel_size[1] // 2
        self.bias = bias
        self.conv = nn.Conv2d(in_channels=self.input_dim + self.hidden_dim,
                              out_channels=4 * self.hidden_dim,
                              kernel_size=self.kernel_size,
                              padding=self.padding,
                              bias=self.bias)

    def forward(self, input_tensor, cur_state):
        h_cur, c_cur = cur_state
        combined = torch.cat([input_tensor, h_cur], dim=1)  # concatenate along channel axis
        combined_conv = self.conv(combined)
        cc_i, cc_f, cc_o, cc_g = torch.split(combined_conv, self.hidden_dim, dim=1)
        i = torch.sigmoid(cc_i)
        f = torch.sigmoid(cc_f)
        o = torch.sigmoid(cc_o)
        g = torch.tanh(cc_g)
        c_next = f * c_cur + i * g
        h_next = o * torch.tanh(c_next)
        return h_next, c_next

    def init_hidden(self, batch_size):
        return (torch.zeros(batch_size, self.hidden_dim, self.height, self.width).cuda(),
                torch.zeros(batch_size, self.hidden_dim, self.height, self.width).cuda())

class ConvLSTM(nn.Module):
    def __init__(self, input_size, input_dim, hidden_dim, kernel_size, num_layers,
                 batch_first=False, bias=True, return_all_layers=False):
        super(ConvLSTM, self).__init__()
        self._check_kernel_size_consistency(kernel_size)
        # Make sure that both `kernel_size` and `hidden_dim` are lists having len == num_layers
        kernel_size = self._extend_for_multilayer(kernel_size, num_layers)
        hidden_dim = self._extend_for_multilayer(hidden_dim, num_layers)
        if not len(kernel_size) == len(hidden_dim) == num_layers:
            raise ValueError('Inconsistent list length.')
        self.height, self.width = input_size
        self.input_dim = input_dim
        self.hidden_dim = hidden_dim
        self.kernel_size = kernel_size
        self.num_layers = num_layers
        self.batch_first = batch_first
        self.bias = bias
        self.return_all_layers = return_all_layers
        cell_list = []
        for i in range(0, self.num_layers):
            cur_input_dim = self.input_dim if i == 0 else self.hidden_dim[i - 1]
            cell_list.append(ConvLSTMCell(input_size=(self.height, self.width),
                                          input_dim=cur_input_dim,
                                          hidden_dim=self.hidden_dim[i],
```

```
                                      kernel_size=self.kernel_size[i],
                                      bias=self.bias))
        self.cell_list = nn.ModuleList(cell_list)
    def forward(self, input_tensor, hidden_state=None):
        """

        Parameters
        ----------
        input_tensor: todo
            5-D Tensor either of shape (t, b, c, h, w) or (b, t, c, h, w)
        hidden_state: todo
            None. todo implement stateful

        Returns
        -------
        last_state_list, layer_output
        """
        if not self.batch_first:
            # (t, b, c, h, w) -> (b, t, c, h, w)
            input_tensor=input_tensor.permute(1, 0, 2, 3, 4)

        # Implement stateful ConvLSTM
        if hidden_state is not None:
            raise NotImplementedError()
        else:
            hidden_state = self._init_hidden(batch_size=input_tensor.size(0))

        layer_output_list = []
        last_state_list = []

        seq_len = input_tensor.size(1)
        cur_layer_input = input_tensor

        for layer_idx in range(self.num_layers):

            h, c = hidden_state[layer_idx]
            output_inner = []
            for t in range(seq_len):
                h, c = self.cell_list[layer_idx](input_tensor=cur_layer_input[:, t, :, :, :],
                                                 cur_state=[h, c])

                output_inner.append(h)

            layer_output = torch.stack(output_inner, dim=1)
            cur_layer_input = layer_output

            layer_output = layer_output.permute(1, 0, 2, 3, 4)

            layer_output_list.append(layer_output)
            last_state_list.append([h, c])

        if not self.return_all_layers:
            layer_output_list = layer_output_list[-1:]
            last_state_list = last_state_list[-1:]

        return layer_output_list, last_state_list

    def _init_hidden(self, batch_size):
        init_states = []
        for i in range(self.num_layers):
            init_states.append(self.cell_list[i].init_hidden(batch_size))
        return init_states

    @staticmethod
    def _check_kernel_size_consistency(kernel_size):
        if not (isinstance(kernel_size, tuple) or
                (isinstance(kernel_size, list) and all([isinstance(elem, tuple) for elem in
kernel_size]))):
            raise ValueError('`kernel_size` must be tuple or list of tuples')

    @staticmethod
    def _extend_for_multilayer(param, num_layers):
```

```
        if not isinstance(param, list):
            param = [param] * num_layers
        return param
```

Program inisialisasi model SegNet-ConvLSTM

```
import torch
import torch.nn as nn
from torchvision import models
import torch.nn.functional as F

class SegNet_ConvLSTM(nn.Module):
    def __init__(self):
        super(SegNet_ConvLSTM,self).__init__()
        self.vgg16_bn = models.vgg16_bn(pretrained=True).features
        self.relu = nn.ReLU(inplace=True)
        self.index_MaxPool = nn.MaxPool2d(kernel_size=2, stride=2,return_indices=True)
        self.index_UnPool = nn.MaxUnpool2d(kernel_size=2, stride=2)
        # net struct
        self.conv1_block = nn.Sequential(self.vgg16_bn[0],
                                         self.vgg16_bn[1],
                                         self.vgg16_bn[2],
                                         self.vgg16_bn[3],
                                         self.vgg16_bn[4],
                                         self.vgg16_bn[5]
                                         )
        self.conv2_block = nn.Sequential(self.vgg16_bn[7],
                                         self.vgg16_bn[8],
                                         self.vgg16_bn[9],
                                         self.vgg16_bn[10],
                                         self.vgg16_bn[11],
                                         self.vgg16_bn[12]
                                         )
        self.conv3_block = nn.Sequential(self.vgg16_bn[14],
                                         self.vgg16_bn[15],
                                         self.vgg16_bn[16],
                                         self.vgg16_bn[17],
                                         self.vgg16_bn[18],
                                         self.vgg16_bn[19],
                                         self.vgg16_bn[20],
                                         self.vgg16_bn[21],
                                         self.vgg16_bn[22]
                                         )
        self.conv4_block = nn.Sequential(self.vgg16_bn[24],
                                         self.vgg16_bn[25],
                                         self.vgg16_bn[26],
                                         self.vgg16_bn[27],
                                         self.vgg16_bn[28],
                                         self.vgg16_bn[29],
                                         self.vgg16_bn[30],
                                         self.vgg16_bn[31],
                                         self.vgg16_bn[32]
                                         )
        self.conv5_block = nn.Sequential(self.vgg16_bn[34],
                                         self.vgg16_bn[35],
                                         self.vgg16_bn[36],
                                         self.vgg16_bn[37],
                                         self.vgg16_bn[38],
                                         self.vgg16_bn[39],
                                         self.vgg16_bn[40],
                                         self.vgg16_bn[41],
                                         self.vgg16_bn[42]
                                         )

        self.upconv5_block = nn.Sequential(
                              nn.Conv2d(512, 512, kernel_size=(3, 3), padding=(1,1)),
                              nn.BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True),self.relu,
                              nn.Conv2d(512, 512, kernel_size=(3, 3), padding=(1,1)),
```

```
                            nn.BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True),self.relu,
                            nn.Conv2d(512, 512, kernel_size=(3, 3), padding=(1, 1)),
                           nn.BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True),self.relu,
                            )
        self.upconv4_block = nn.Sequential(
                            nn.Conv2d(512, 512, kernel_size=(3, 3), padding=(1,1)),
                            nn.BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True),self.relu,
                            nn.Conv2d(512, 512, kernel_size=(3, 3), padding=(1,1)),
                            nn.BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True),self.relu,
                            nn.Conv2d(512, 256, kernel_size=(3, 3), padding=(1, 1)),
                            nn.BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True),self.relu,
                                        )
        self.upconv3_block = nn.Sequential(
                            nn.Conv2d(256, 256, kernel_size=(3, 3), padding=(1,1)),
                            nn.BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True),self.relu,
                            nn.Conv2d(256, 256, kernel_size=(3, 3), padding=(1,1)),
                            nn.BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True),self.relu,
                            nn.Conv2d(256, 128, kernel_size=(3, 3), padding=(1, 1)),
                            nn.BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True),self.relu,
                                        )
        self.upconv2_block = nn.Sequential(
                            nn.Conv2d(128, 128, kernel_size=(3, 3), padding=(1,1)),
                            nn.BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True),self.relu,
                            nn.Conv2d(128, 64, kernel_size=(3, 3), padding=(1,1)),
                            nn.BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True),self.relu
                                        )
        self.upconv1_block = nn.Sequential(
                            nn.Conv2d(64, 64, kernel_size=(3, 3), padding=(1,1)),
                            nn.BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True),self.relu,
                            nn.Conv2d(64, 2, kernel_size=(3, 3), padding=(1,1)),
                                        )
        self.convlstm = ConvLSTM(input_size=(4,8),
                                 input_dim=512,
                                 hidden_dim=[512, 512],
                                 kernel_size=(3,3),
                                 num_layers=2,
                                 batch_first=False,
                                 bias=True,
                                 return_all_layers=False)
    def forward(self, x):
        x = torch.unbind(x, dim=1)
        data = []
        for item in x:
            f1, idx1 = self.index_MaxPool(self.conv1_block(item))
            f2, idx2 = self.index_MaxPool(self.conv2_block(f1))
            f3, idx3 = self.index_MaxPool(self.conv3_block(f2))
            f4, idx4 = self.index_MaxPool(self.conv4_block(f3))
            f5, idx5 = self.index_MaxPool(self.conv5_block(f4))
            data.append(f5.unsqueeze(0))
        data = torch.cat(data, dim=0)
        lstm, _ = self.convlstm(data)
        test = lstm[0][-1,:,:,:,:]
        up6 = self.index_UnPool(test,idx5)
        up5 = self.index_UnPool(self.upconv5_block(up6), idx4)
        up4 = self.index_UnPool(self.upconv4_block(up5), idx3)
        up3 = self.index_UnPool(self.upconv3_block(up4), idx2)
        up2 = self.index_UnPool(self.upconv2_block(up3), idx1)
        up1 = self.upconv1_block(up2)
        return F.log_softmax(up1, dim=1)
```

Program inisialisasi model UNet-ConvLSTM

```
class UNet_ConvLSTM(nn.Module):
    def __init__(self, n_channels, n_classes):
        super(UNet_ConvLSTM, self).__init__()
        self.inc = inconv(n_channels, 64)
        self.down1 = down(64, 128)
        self.down2 = down(128, 256)
```

```
        self.down3 = down(256, 512)
        self.down4 = down(512, 512)
        self.up1 = up(1024, 256)
        self.up2 = up(512, 128)
        self.up3 = up(256, 64)
        self.up4 = up(128, 64)
        self.outc = outconv(64, n_classes)
        self.convlstm = ConvLSTM(input_size=(8,16),
                                 input_dim=512,
                                 hidden_dim=[512, 512],
                                 kernel_size=(3,3),
                                 num_layers=2,
                                 batch_first=False,
                                 bias=True,
                                 return_all_layers=False)

    def forward(self, x):
        x = torch.unbind(x, dim=1)
        data = []
        for item in x:
            x1 = self.inc(item)
            x2 = self.down1(x1)
            x3 = self.down2(x2)
            x4 = self.down3(x3)
            x5 = self.down4(x4)
            data.append(x5.unsqueeze(0))
        data = torch.cat(data, dim=0)
        lstm, _ = self.convlstm(data)
        test = lstm[0][ -1,:, :, :, :]
        x = self.up1(test, x4)
        x = self.up2(x, x3)
        x = self.up3(x, x2)
        x = self.up4(x, x1)
        x = self.outc(x)
        return x, test
```

Program untuk menjalankan *training* dan *validation* untuk model

```
from torch.utils.data import Dataset
from PIL import Image
import torch
#import config
import torchvision.transforms as transforms
import numpy as np
from sklearn import preprocessing

def readTxt(file_path):
    img_list = []
    with open(file_path, 'r') as file_to_read:
        while True:
            lines = file_to_read.readline()
            if not lines:
                break
            item = lines.strip().split()
            img_list.append(item)
    file_to_read.close()
    return img_list

class RoadSequenceDatasetList(Dataset):

    def __init__(self, file_path, transforms):

        self.img_list = readTxt(file_path)
        self.dataset_size = len(self.img_list)
        self.transforms = transforms
    def __len__(self):
        return self.dataset_size
```

```
    def __getitem__(self, idx):
        img_path_list = self.img_list[idx]
        data = []
        for i in range(5):
            data.append(torch.unsqueeze(self.transforms(Image.open(img_path_list[i])), dim=0))
        data = torch.cat(data, 0)
        label = Image.open(img_path_list[5])
        label = torch.squeeze(self.transforms(label))
        sample = {'data': data, 'label': label}
        return sample

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
train_path    =    'D:/Kuliah/SEMESTER    8/Robust-Lane-Detection-master/Robust-Lane-Detection-
master/LaneDetectionCode/data/train_index.txt'
val_path     =     'D:/Kuliah/SEMESTER     8/Robust-Lane-Detection-master/Robust-Lane-Detection-
master/LaneDetectionCode/data/DatasetList.txt'
transform = transforms.Compose([transforms.ToTensor()])
batch_size = 4;
from torch.utils.data import DataLoader
train_set = RoadSequenceDatasetList(file_path=train_path,transforms=transform)
trainloader = DataLoader(train_set,batch_size=batch_size,shuffle=True)
val_set = RoadSequenceDatasetList(file_path=val_path,transforms=transform)
val_loader = DataLoader(val_set,batch_size=batch_size,shuffle=True)

from torch import optim
import torch
#from jcopdl.callback import Callback, set_config
from _callback import CallbackSemSeg
from _config import set_config
learning_rate = 0.0001

path     =     'D:/Kuliah/DeepLearning/Code/SemanticSegmentation/4_SemanticSegmentationTry/Tugas
Akhir/3_TA_UNetConvLSTM/Epoch51/weights_best.pth'
model = UNet_ConvLSTM(3,2).to(device)
model.load_state_dict(torch.load(path))
criterion = nn.CrossEntropyLoss(weight=torch.Tensor([0.02, 2.02]).to(device))

from tqdm.auto import tqdm
import time
def loop_fn(mode,dataloader,model,criterion,optimizer,device):
    if mode == 'train':
        model.train()
    elif mode == 'val':
        model.eval()
    cost = 0
    acc = 0
    prec = 0
    rec = 0
    F1 = 0
    IoU = 0
    for sample_batched in tqdm(dataloader, desc=mode.title()):
       data,target=sample_batched['data'].to(device),
       sample_batched['label'].type(torch.LongTensor).to(device) # LongTensor
       optimizer.zero_grad()
       output,_ = model(data)
       output = output.type(torch.Tensor).to(device)
       loss = criterion(output, target).to(device)
       if mode == 'train':
            loss.backward()
            optimizer.step()
            metric_calculator=SegmentationMetrics(average=True,ignore_background=False,
                                                  activation='softmax')
        cost += criterion(output, target).item()
        pixel_accuracy, F1_score, precision, recall,IoU_score = metric_calculator(target, output)
        acc += pixel_accuracy.item()
        prec += precision.item()
        rec += recall.item()
        F1 += F1_score.item()
        IoU += IoU_score.item()

    cost /= (len(dataloader.dataset)/batch_size)
    acc  /= (len(dataloader.dataset)/batch_size)
```

```
    prec /= (len(dataloader.dataset)/batch_size)
    rec  /= (len(dataloader.dataset)/batch_size)
    F1   /= (len(dataloader.dataset)/batch_size)
    IoU  /= (len(dataloader.dataset)/batch_size)
    return cost,acc,F1,prec,rec,IoU
```

Program Metrik Performa

```
import numpy as np
import torch
import torch.nn as nn
from torchvision import transforms
from PIL import Image

class SegmentationMetrics(object):
    r"""Calculate common metrics in semantic segmentation to evalueate model preformance.
    Supported metrics: Pixel accuracy, Dice Coeff, precision score and recall score.

    Pixel accuracy measures how many pixels in a image are predicted correctly.
    Dice Coeff is a measure function to measure similarity over 2 sets, which is usually used to
    calculate the similarity of two samples. Dice equals to f1 score in semantic segmentation tasks.

    It should be noted that Dice Coeff and Intersection over Union are highly related, so you need
    NOT calculate these metrics both, the other can be calcultaed directly when knowing one of them.
    Precision describes the purity of our positive detections relative to the ground truth. Of all
     the objects that we predicted in a given image, precision score describes how many of those
objects
    actually had a matching ground truth annotation.
    Recall describes the completeness of our positive predictions relative to the ground truth. Of
    all the objected annotated in our ground truth, recall score describes how many true positive
instances
    we have captured in semantic segmentation.
    Args:
        eps: float, a value added to the denominator for numerical stability.
            Default: 1e-5
        average: bool. Default: ``True``
            When set to ``True``, average Dice Coeff, precision and recall are
            returned. Otherwise Dice Coeff, precision and recall of each class
            will be returned as a numpy array.
        ignore_background: bool. Default: ``True``
            When set to ``True``, the class will not calculate related metrics on
            background pixels. When the segmentation of background pixels is not
            important, set this value to ``True``.
        activation: [None, 'none', 'softmax' (default), 'sigmoid', '0-1']
            This parameter determines what kind of activation function that will be
            applied on model output.
    Input:
        y_true: :math:`(N, H, W)`, torch tensor, where we use int value between (0, num_class - 1)
        to denote every class, where ``0`` denotes background class.
        y_pred: :math:`(N, C, H, W)`, torch tensor.
    Examples::
        >>> metric_calculator = SegmentationMetrics(average=True, ignore_background=True)
        >>> pixel_accuracy, dice, precision, recall = metric_calculator(y_true, y_pred)
    """
    def __init__(self, eps=1e-5, average=True, ignore_background=True, activation='0-1'):
        self.eps = eps
        self.average = average
        self.ignore = ignore_background
        self.activation = activation

    @staticmethod
    def _one_hot(gt, pred, class_num):
        # transform sparse mask into one-hot mask
        # shape: (B, H, W) -> (B, C, H, W)
        input_shape = tuple(gt.shape)  # (N, H, W, ...)
        new_shape = (input_shape[0], class_num) + input_shape[1:]
        one_hot = torch.zeros(new_shape).to(pred.device, dtype=torch.float)
```

```
        target = one_hot.scatter_(1, gt.unsqueeze(1).long().data, 1.0)
        return target

    @staticmethod
    def _get_class_data(gt_onehot, pred, class_num):
        # perform calculation on a batch
        # for precise result in a single image, plz set batch size to 1
        matrix = np.zeros((3, class_num))

        # calculate tp, fp, fn per class
        for i in range(class_num):
            # pred shape: (N, H, W)
            class_pred = pred[:, i, :, :]
            #print(class_pred.size())
            # gt shape: (N, H, W), binary array where 0 denotes negative and 1 denotes positive
            class_gt = gt_onehot[:, i, :, :]
            class_gt = transforms.Grayscale(num_output_channels=1)(class_gt)
            #print(class_gt.size())
            pred_flat = class_pred.contiguous().view(-1, )  # shape: (N * H * W, )
            #print(pred_flat.size())
            gt_flat = class_gt.contiguous().view(-1, )  # shape: (N * H * W, )
            #print(gt_flat.size())

            tp = torch.sum(gt_flat * pred_flat)
            fp = torch.sum(pred_flat) - tp
            fn = torch.sum(gt_flat) - tp

            matrix[:, i] = tp.item(), fp.item(), fn.item()

        return matrix

    def _calculate_multi_metrics(self, gt, pred, class_num):
        # calculate metrics in multi-class segmentation
        matrix = self._get_class_data(gt, pred, class_num)
        if self.ignore:
            matrix = matrix[:, 1:]

        # tp = np.sum(matrix[0, :])
        # fp = np.sum(matrix[1, :])
        # fn = np.sum(matrix[2, :])

       pixel_acc = (np.sum(matrix[0, :]) + self.eps) / (np.sum(matrix[0, :]) + np.sum(matrix[1, :]))
        dice = (2 * matrix[0] + self.eps) / (2 * matrix[0] + matrix[1] + matrix[2] + self.eps)
        precision = (matrix[0] + self.eps) / (matrix[0] + matrix[1] + self.eps)
        recall = (matrix[0] + self.eps) / (matrix[0] + matrix[2] + self.eps)
        IoU = (matrix[0] + self.eps) / (matrix[0] + matrix[1] + matrix[2] + self.eps)

        if self.average:
            dice = np.average(dice)
            precision = np.average(precision)
            recall = np.average(recall)
            IoU = np.average(IoU)

        return pixel_acc, dice, precision, recall, IoU

    def __call__(self, y_true, y_pred):
        class_num = y_pred.size(1)

        if self.activation in [None, 'none']:
            activation_fn = lambda x: x
            activated_pred = activation_fn(y_pred)
        elif self.activation == "sigmoid":
            activation_fn = nn.Sigmoid()
            activated_pred = activation_fn(y_pred)
        elif self.activation == "softmax":
            activation_fn = nn.Softmax(dim=1)
            activated_pred = activation_fn(y_pred)
        elif self.activation == "0-1":
            pred_argmax = torch.argmax(y_pred, dim=1)
            activated_pred = self._one_hot(pred_argmax, y_pred, class_num)
        else:
            raise NotImplementedError("Not a supported activation!")
```

```
        gt_onehot = self._one_hot(y_true, y_pred, class_num)
           pixel_acc, dice, precision, recall, IoU = self._calculate_multi_metrics(gt_onehot,
activated_pred, class_num)
        return pixel_acc, dice, precision, recall, IoU

class BinaryMetrics():
    r"""Calculate common metrics in binary cases.
    In binary cases it should be noted that y_pred shape shall be like (N, 1, H, W), or an assertion
    error will be raised.
    Also this calculator provides the function to calculate specificity, also known as true negative
    rate, as specificity/TPR is meaningless in multiclass cases.
    """
    def __init__(self, eps=1e-5, activation='0-1'):
        self.eps = eps
        self.activation = activation

    def _calculate_overlap_metrics(self, gt, pred):
        output = pred.view(-1, )
        target = gt.view(-1, ).float()

        tp = torch.sum(output * target)  # TP
        fp = torch.sum(output * (1 - target))  # FP
        fn = torch.sum((1 - output) * target)  # FN
        tn = torch.sum((1 - output) * (1 - target))  # TN

        pixel_acc = (tp + tn + self.eps) / (tp + tn + fp + fn + self.eps)
        dice = (2 * tp + self.eps) / (2 * tp + fp + fn + self.eps)
        precision = (tp + self.eps) / (tp + fp + self.eps)
        recall = (tp + self.eps) / (tp + fn + self.eps)
        specificity = (tn + self.eps) / (tn + fp + self.eps)

        return pixel_acc, dice, precision, specificity, recall

    def __call__(self, y_true, y_pred):
        # y_true: (N, H, W)
        # y_pred: (N, 1, H, W)
        if self.activation in [None, 'none']:
            activation_fn = lambda x: x
            activated_pred = activation_fn(y_pred)
        elif self.activation == "sigmoid":
            activation_fn = nn.Sigmoid()
            activated_pred = activation_fn(y_pred)
        elif self.activation == "0-1":
            sigmoid_pred = nn.Sigmoid()(y_pred)
            activated_pred = (sigmoid_pred > 0.5).float().to(y_pred.device)
        else:
            raise NotImplementedError("Not a supported activation!")

        assert activated_pred.shape[1] == 1, 'Predictions must contain only one channel' \
                                             ' when performing binary segmentation'
 pixel_acc,dice,precision,specificity,recall=self._calculate_overlap_metrics(y_true.to(y_pred.devi
                                             ce,dtype=torch.float),activated_pred)
 return [pixel_acc, dice, precision, specificity, recall]
```

Program Deteksi Lajur dari masukan video

```
import cv2
import os
from torchvision import transforms
import torchvision
from PIL import Image
convert_tensor = transforms.ToTensor()
cap = cv2.VideoCapture('/content/drive/MyDrive/VideoRaw/MalamKiri.mp4')
try:
  if not os.path.exists('/content/drive/MyDrive/Video Processing_SegNetLSTM/MalamKiri/'):
    os.makedirs('/content/drive/MyDrive/Video Processing_SegNetLSTM/MalamKiri/')
```

```
except OSError:
  print('Error: Creating directory of data')

currentframe = 1
while(True):
  success,frame = cap.read()
  if success:
    path = '/content/drive/MyDrive/Video Processing_SegNetLSTM/MalamKiri/'
    name = path + 'frame' + str(currentframe) +'.jpg'
    pred_name = path + 'pred' + str(currentframe) + '.jpg'
    print('creating...'+name)

    with torch.no_grad():
      if currentframe % 5 == 0:
        model.eval()
        frame = cv2.resize(frame,(256,128))
        cv2.imwrite(name, frame)
        list_image = getList(path)
        #print(list_image)
        data = []
        data1 = torch.unsqueeze(convert_tensor(Image.open(list_image[-5])),0)
        data.append(data1)
        data2 = torch.unsqueeze(convert_tensor(Image.open(list_image[-4])),0)
        data.append(data2)
        data3 = torch.unsqueeze(convert_tensor(Image.open(list_image[-3])),0)
        data.append(data3)
        data4 = torch.unsqueeze(convert_tensor(Image.open(list_image[-2])),0)
        data.append(data4)
        data5 = torch.unsqueeze(convert_tensor(Image.open(list_image[-1])),0)
        data.append(data5)

        data = torch.cat(data,0).unsqueeze(0).to(device)
        #print(data)
        pred = model(data)
        pred = pred.max(1,keepdim=True)[1].type(torch.Tensor)
        torchvision.utils.save_image(pred,pred_name)

      frame = cv2.resize(frame,(256,128))
      cv2.imwrite(name, frame)
      currentframe +=1
  else:
    break

cap.release()
cv2.destroyAllWindows()
```

Halaman ini sengaja dikosongkan

BIODATA PENULIS

Penulis dilahirkan di Malang, 1 Juni 2001, merupakan anak terakhir dari 3 bersaudara. Penulis telah menempuh pendidikan formal yaitu di MIN 1 Kota Malang, MTsN 1 Kota Malang dan MAN 2 Kota Malang. Setelah lulus dari SMA tahun 2018, Penulis mengikuti SBMPTN dan diterima di Departemen Teknik Elektro FTEIC - ITS pada tahun 2018 dan terdaftar dengan NRP 07111840000077.

Di Departemen Teknik Elektro Penulis sempat aktif di beberapa kegiatan seperti Program Kreativitas Mahasiswa dan aktif sebagai Asisten Dosen pada mata kuliah Matematika 1 dan Matematika 2.