IMPLEMENTASI DEEP LEARNING UNTUK PENDETEKSIAN KANTUK BERDASARKAN FITUR KEDIPAN



Disusun Oleh:

Nama : Hanif Annur Rahman

NIM : 13523090

PROGRAM STUDI INFORMATIKA – PROGRAM SARJANA

FAKULTAS TEKNOLOGI INDUSTRI

UNIVERSITAS ISLAM INDONESIA

2020

HALAMAN PENGESAHAN DOSEN PEMBIMBING

IMPLEMENTASI DEEP LEARNING UNTUK PENDETEKSIAN KANTUK BERDASARKAN FITUR KEDIPAN

TUGAS AKHIR

Disusun Oleh:

Nama : Hanif Annur Rahman

NIM : 13523090

Yogyakarta, 14 Oktober 2020

Pembimbing,

(Arrie Kurniawardhani, S.Si., M.Kom.)

HALAMAN PENGESAHAN DOSEN PENGUJI

IMPLEMENTASI DEEP LEARNING UNTUK PENDETEKSIAN KANTUK BERDASARKAN FITUR KEDIPAN

TUGAS AKHIR

Telah dipertahankan di depan sidang penguji sebagai salah satu syarat untuk memperoleh gelar Sarjana Komputer dari Program Studi Informatika di Fakultas Teknologi Industri Universitas Islam Indonesia

Yogyakarta, 14 Oktober 2020

Tim Penguji

Arrie Kurniawardhani, S.Si., M.Kom. _____

Anggota 1

Dhomas Hatta Fudholi, S.T., M.Eng., Ph.D. _____

Anggota 2

Syarif Hidayat, Dr., S.Kom., M.I.T. _____

Mengetahui,

Ketua Program Studi Informatika – Program Sarjana

Fakultas Teknologi Industri

Universitas Islam Indonesia

(Dr. Raden Teduh Dirgahayu, S.T., M.Sc.)

HALAMAN PERNYATAAN KEASLIAN TUGAS AKHIR

Yang bertanda tangan di bawah ini:

Nama : Hanif Annur Rahman

NIM : 13523090

Tugas akhir dengan judul:

IMPLEMENTASI DEEP LEARNING UNTUK PENDETEKSIAN KANTUK BERDASARKAN FITUR KEDIPAN

Menyatakan bahwa seluruh komponen dan isi dalam tugas akhir ini adalah hasil karya saya sendiri. Apabila dikemudian hari terbukti ada beberapa bagian dari karya ini adalah bukan hasil karya sendiri, tugas akhir yang diajukan sebagai hasil karya sendiri ini siap ditarik kembali dan siap menanggung resiko dan konsekuensi apapun.

Demikian surat pernyataan ini dibuat, semoga dapat dipergunakan sebagaimana mestinya.

Yogyakarta, 14 Oktober 2020



(Hanif Annur Rahman)

HALAMAN PERSEMBAHAN

Bismillahirahmanirrahim,

Syukur alhamdulillah atas izin Allah Subhanahu wa Ta'ala akhirnya saya dapat menyelesaikan tugas akhir ini dengan baik.

Terimakasih yang tak terhingga saya ucapkan kepada Ayah dan Ibu yang selalu mendoakan saya dan selalu memberi dukungan kepada saya selama pengerjaan tugas akhir. Dengan laporan inilah salah bentuk wujud terimakasih saya dan berharap semoga mampu membuat bahagia dan bangga kepada anaknya.

Penulis juga mengucapkan terimakasih sebanyak-banyaknya kepada semua pihak yang telah membantu, mendo'akan, dan mendukung saya selama pengerjaan tugas akhir ini. Semoga Allah Subhanahu wa Ta'ala selalu memberi keberkahan, kesehatan dan rezeki kepada kita semua, Aamiin

HALAMAN MOTO

"*Sebaik-baik manusia adalah yang paling bermanfaat bagi manusia*"

(HR. Ahmad)

KATA PENGANTAR

Assalaamu'alaikum warahmatullahi wabarakatuh

Puji dan syukur diucapkan atas kehadirat Allah Subhanahu wa ta'alaa yang telah memberikan berkat Rahmat dan hidayah-Nya sehingga dapat menyelesaikan laporan ini. Shalawat serta salam senantiasa dilimpahkan kepada Nabi Muhammad Shallallahu 'alaihi wasallam, sebagai pembawa risalah Allah terakhir dan penyempurna seluruh risalah-Nya.

Izinkanlah untuk menyampaikan terima kasih kepada semua pihak yang telah berjasa memberikan ide dan motivasi dalam rangka menyelesaikan laporan ini:

1. Allah Subhanahu wa ta'alaa yang telah memberikan kemudahan dalam membuat laporan ini.
2. Orang tua yang selalu memberikan do'a dan motivasi sehingga dapat menyelesaikan laporan ini.
3. Bapak Fathul Wahid, S.T., M.Sc., Ph.D selaku Rektor Universitas Islam Indonesia
4. Bapak Hari Purnomo, Prof., Dr., Ir., M.T. selaku Dekan Fakultas Teknologi Industri, Universitas Islam Indonesia.
5. Bapak Dr. Raden Teguh, S.T., M.Sc., selaku Ketua Program Studi Informatika – Program Sarjana, Fakultas Teknologi Industri, Universitas Islam Indonesia.
6. Arrie Kurniawardhani, S.Si., M.Kom., selaku dosen pembimbing tugas akhir yang telah memberikan bimbingan dan motivasinya selama ini.
7. Seluruh staf pengajar di Jurusan Informatika, Fakultas Teknologi Industri, Universitas Islam Indonesia.
8. Staf administrasi FTI, secara khusus kepada staf administrasi Program Studi Informatika – Program Sarjana, Fakultas Teknologi Industri, Universitas Islam Indonesia yang selalu memberika informasi terkait tugas akhir.
9. Semua pihak yang tidak dapat penulis sebutkan satu persatu yang juga trut membantu penulis selama pengerjaan tugas akhir.
10. Rahmad Arifan Hr dan Rickoh Andi Muchtar yang telah membantu memberikan solusi ketika penulis mengalami kendala selama pengerjaan tugas akhir.

Semoga kebaikan yang diberikan oleh semua pihak dapat menjadi amal shaleh. Tidak ada manusia yang luput dari kesalahan sehingga memungkinkan terdapat kesalahan dalam laporan ini, maka dari itu saran dan kritik yang sifatnya membangun akan diterima dengan senang hati.

Pada akhirnya bukakan pintu maaf yang sebesar-besarnya apabila laporan ini memiliki kekurangan, baik yang disengaja maupun tidak disengaja.

Wassalaamu'alaikum warahmatullaahi wabarakatuh

Yogyakarta, 14 Oktober 2020



(Hanif Annur Rahman)

SARI

Kantuk adalah salah satu faktor penyebab terjadinya kecelakaan lalu lintas. Menurut Badan Pengatur Jalan Tol, kantuk menyumbang 33% dari seluruh penyebab kecelakaan yang terjadi di jalan tol trans jawa pada bulan Desember 2018 sampai Februari 2019. Oleh karena itu diperlukannya solusi untuk mengatasi masalah kantuk tersebut. Salah satu solusi untuk mengatasi masalah kantuk adalah dengan memberikan peringatan melalui suara alarm. Salah satu cara untuk menyelesaikan masalah tersebut menggunakan metode *deep learning* yang dikembang oleh Shuali Li et al. yaitu *Independently Recurrent Neural Network* (IndRNN).

Untuk menyelesaikan masalah tersebut dilakukan dengan dua langkah yaitu pengumpulan data dan melakukan pendeteksian kantuk. Untuk melakukan pendeteksian kantuk dilakukan dengan tujuh tahap yaitu pendeteksian wajah, pendeteksian *facial landmark*, penghitungan EAR, pendeteksian kedipan, penghitungan 4 fitur kedipan, normalisasi, dan pengklasifikasian dengan IndRNN.

Hasil dari pengimplementasian IndRNN tersebut kemudian dibandingkan dengan *Hierarchical Multiscale Long Short-Term Memory* (HM-LSTM) yang menunjukkan bahwa IndRNN memiliki akurasi lebih baik saat diuji, dengan hasil akurasi 91,88% untuk IndRNN dan 86,48% untuk HM-LSTM. Waktu yang dibutuhkan IndRNN untuk *training* dengan 80% *dataset* UTA-RLDD yaitu 53 detik, lebih cepat dibandingkan dengan HM-LSTM yang membutuhkan waktu 670 detik untuk *training* dengan *dataset* yang sama.

Kata kunci: *Deep Learning, Independently Recurrent Neural Network, Hierarchical Multiscale Long Short-Term Memory, Eye Aspect Ratio*

GLOSARIUM

Array	Struktur data yang menyimpan banyak elemen yang bertipe sama dan mempunyai ukuran yang tetap
Dataset	Data yang akan diolah
Decode	Proses menerjemahkan suatu informasi agar mudah dimengerti
Encode	Proses pengubahan informasi menjadi ke dalam bentuk yang lebih efisien untuk dikirim atau disimpan
Facial Landmark	Titik-titik pada wajah yang dapat merepresentasikan atau mendeskripsikan suatu wajah
Frame	Satuan gambar pada suatu video
Grayscale	Warna-warna monokromatik dari hitam menjadi putih
Layer	Sebuah komponen tingkat tinggi dalam pembentukan *deep learning*
Mask	Suatu operasi yang dilakukan pada suatu gambar
Overfitting	Model yang terlalu mempelajari data *training*
Preprocessing	Proses yang mengubah suatu data, agar data tersebut siap dipakai oleh program lain
Training	Memberikan data ke *machine learning* agar *machine learning* tersebut mendapatkan pengetahuan berupa pola yang tergeneralisasi yang dapat digunakan untuk klasifikasi atau prediksi

DAFTAR ISI

HALAMAN PENGESAHAN DOSEN PEMBIMBING ii
HALAMAN PENGESAHAN DOSEN PENGUJI iii
HALAMAN PERNYATAAN KEASLIAN TUGAS AKHIR iv
HALAMAN PERSEMBAHAN v
HALAMAN MOTO vi
KATA PENGANTAR vii
SARI ix
GLOSARIUM x
DAFTAR ISI xi
DAFTAR TABEL xiii
DAFTAR GAMBAR xiv
BAB I PENDAHULUAN 1
1.1 Latar Belakang 1
1.2 Rumusan Masalah 2
1.3 Batasan Masalah 2
1.4 Tujuan Penelitian 2
1.5 Manfaat Penelitian 3
1.6 Sistematika Penulisan 3
BAB II TINJAUAN PUSTAKA 4
BAB III LANDASAN TEORI 6
3.1 Kantuk 6
3.2 Pendeteksian Wajah Dengan HOG (Histogram of Oriented Gradients) 6
3.2.1 Normalisasi 6
3.2.2 Hitung Besaran Gradien dan Orientasi Gradien 7
3.2.3 Orientation Binning 7
3.2.4 Klasifikasi dengan SVM 7
3.3 Pendeteksian Facial Landmark dengan Ensemble of Regression Trees 8
3.4 Support Vector Mahinces (SVM) 8
3.5 Pendeteksian Kedipan dengan Eye Aspect Ratio (EAR) 9
3.6 Algoritma Pengambilan Kedipan 10
3.7 Penghitungan Nilai 4 Fitur Kedipan 11
3.8 Deep Learning 11

3.9 Recurrent Neural Neworks (RNN) 12
3.10 Hierarchical Multiscale Long Short-Term Memory (HM-LSTM) 12
3.11 Independently Recurrent Neural Networks (IndRNN) 13
BAB IV METODOLOGI PENELITIAN 16
4.1 Pengumpulan Data 16
4.2 Alur Pendeteksian Kantuk 18
4.2.1 Pendeteksian Wajah 19
4.2.2 Pendeteksian Facial Landmark 28
4.2.3 Penghitugan EAR 29
4.2.4 Pendeteksian Kedipan 30
4.2.5 Penghitungan 4 Fitur Kedipan 33
4.2.6 Preprocessing 35
4.2.7 Pengklasifikasian dengan IndRNN 35
4.3 Training 36
4.4 Pengujian 38
4.5 Metode yang akan Dibandingkan 38
BAB V HASIL DAN PEMBAHASAN 39
5.1 Implementasi 39
5.1.1 Pendeteksian Wajah 39
5.1.2 Pendeteksian Facial Landmark 41
5.1.3 Penghitungan EAR 41
5.1.4 Pendeteksian Kedipan 43
5.1.5 Penghitungan 4 Fitur Kedipan 45
5.1.6 Preprocessing 45
5.1.7 Pengklasifikasian dengan IndRNN 47
5.2 Training 48
5.3 Pengujian 49
5.4 Perbandingan Hasil Implementasi 51
BAB VI PENUTUP 52
6.1 Kesimpulan 52
6.2 Saran 52
DAFTAR PUSTAKA 53

DAFTAR TABEL

Tabel 2.1 Pebandingan Penelitian Sebelumnya 5
Tabel 5.1 Hasil pengujian model 50

DAFTAR GAMBAR

Gambar 3.1 Ilustrasi bagaimana SVM memisahkan dua kelas dengan *hyperplane* 9
Gambar 3.2 Koordinat *facial landmark* pada mata 10
Gambar 3.3 Arsitektur RNN 12
Gambar 3.4 Hubungan Antar Neuron pada IndRNN 14
Gambar 3.5 Hubungan antar neuron pada RNN biasa 15
Gambar 4.1 Sampel gambar orang tidak mengantuk (baris pertama), orang agak mengantuk (baris kedua), dan orang mengantuk (baris ketiga) 17
Gambar 4.2 *Flowchart* alur pendeteksian kantuk 18
Gambar 4.3 *Centered mask* arah vertikal 19
Gambar 4.4 *Centered mask* arah horizontal 19
Gambar 4.5 Contoh nilai piksel pada suatu gambar berukuran 5x4 piksel 20
Gambar 4.6 Gambar wajah yang sudah dibagi menjadi beberapa *cell* 22
Gambar 4.7 Bin sebanyak 9 sebuah yang dinamai berdasarkan orientasi dari 0-180° 22
Gambar 4.8 Besaran gradien tiap piksel 23
Gambar 4.9 Orientasi gradien tiap piksel 23
Gambar 4.10 Contoh proses *orientation binning* 24
Gambar 4.11 Histogram besaran gradien untuk tiap orientasi gradien 28
Gambar 4.12 Proses *regressor* memprediksi *facial landmark* pada tiap iterasi 29
Gambar 4.13 Koordinat *facial landmark* pada mata 30
Gambar 4.14 Grafik nilai EAR yang terdiri dari 2 kedipan 30
Gambar 4.15 Grafik nilai EAR 33
Gambar 4.16 Arsitektur IndRNN 36
Gambar 4.17 Cara kerja *training* IndRNN 37
Gambar 5.1 Kode *gamma correction* 39
Gambar 5.2 Kode penghitungan besaran dan orientasi gradien serta *orientation binning* 40
Gambar 5.3 Hasil pendeteksian wajah 40
Gambar 5.4 Kode pendeteksian *facial landmark* 41
Gambar 5.5 Hasil pendeteksian *facial landmark* 41
Gambar 5.6 Kode penghitungan EAR 42
Gambar 5.7 Wajah orang yang sedang berkedip 43
Gambar 5.8 Kode penerapan SVM 43
Gambar 5.9 Grafik EAR pada rentang 13 *frame* 43

Gambar 5.10 Kode implementasi algoritma Ghoddoosian et al. 45
Gambar 5.11 Kode penghitungan 4 fitur kedipan 45
Gambar 5.12 Kode implementasi normalisasi data kedipan 46
Gambar 5.13 Kode implementasi pengelompokkan data kedipan 46
Gambar 5.14 Kode implementasi IndRNN 47
Gambar 5.15 Kode implementasi *training* IndRNN 49
Gambar 5.16 Hasil dari penjalanan kode *training* 49

BAB I
PENDAHULUAN

1.1 Latar Belakang

Kantuk merupakan salah satu faktor penyebab terjadinya kecelakaan lalu lintas. Berdasarkan data dari Badan Pengatur Jalan Tol, kantuk menyumbang 33% dari seluruh penyebab kecelakaan yang terjadi di jalan tol trans jawa pada bulan Desember 2018 sampai Februari 2019. Studi epidemiologi kecelakaan lalu lintas yang dilakukan oleh Iskandar Arfan dan Wulandari (2018) di Pontianak, menunjukkan faktor mengantuk menyumbang sebesar 26,6% sebagai penyebab kecelakaan yang terjadi di Pontianak. Fakta-fakta tersebut menunjukkan diperlukannya solusi untuk mengatasi masalah kantuk tersebut.

Salah satu solusi untuk mengatasi masalah kantuk adalah dengan memberikan peringatan melalui suara alarm (Chandrasena & Wickramasinghe, 2016). Untuk membuat sistem yang dapat memberikan peringatan kepada seseorang saat seseorang mengantuk, sistem tersebut perlu memiliki fitur pendeteksi kantuk yang dapat mengetahui seseorang itu mengantuk atau tidak agar dapat bekerja dengan baik. Penelitian yang dilakukan oleh Friedrichs dan Yang (2010) menunjukkan kedipan dapat digunakan untuk menentukan seseorang itu mengantuk atau tidak. Dalam penelitian tersebut, untuk mengklasifikasikan kantuk dilakukan dengan memanfaatkan fitur-fitur kedipan seperti kecepatan, amplitudo, frekuensi, durasi, dan lain-lain Pengklasifikasian tersebut bertujuan untuk menentukan apakah mengantuk atau tidak. Tingkat keakuratan pengklasifikasi tersebut sebesar 82,5%, dengan artian angka tersebut dapat dikatakan cukup akurat. Oleh karena itu kedipan dapat digunakan oleh pendeteksi kantuk sebagai penentu seseorang itu mengantuk atau tidak.

Sebagaimana kita ketahui kedipan seseorang berubah-ubah dalam waktu ke waktu, fitur-fitur kedipan ini termasuk ke dalam data temporal, maka pendeteksi kantuk harus mengimplementasi metode yang dapat mengklasifikasi data temporal. Semenjak *deep learning* menjadi populer setelah variasi arsitektur *Convolutional Neural Network* (CNN) bernama AlexNet menjadi pemenang *ImageNet Large Scale Visual Recognition Challenge* (ILSVRC) pada tahun 2012, *deep learning* tidak hanya digunakan untuk mengenali gambar lagi tetapi digunakan untuk mengatasi masalah lainnya, salah satunya adalah klasifikasi data temporal. Dalam masalah pengenalan aktivitas manusia, aktivitas manusia bisa dikenali dari data sensor seperti barometer dan akselerometer yang datanya termasuk ke dalam data temporal. Penelitian tentang penerapan *deep learning* untuk pengenalan aktivitas manusia berdasarkan sensor

akselerometer yang dilakukan oleh Pienaar dan Malekian (2019), menunjukkan *deep learning* dapat digunakan untuk mengenali aktivitas manusia yang berarti *deep learning* dapat digunakan untuk mengklasifkasi data yang berbentuk temporal.

Pada tahun 2019, Shuali Li et al. mengembangkan *Independently Recurrent Neural Network* (IndRNN) yang variasi *deep learning* RNN. Berdasarkan penelitian Shuali Li et al. (2019), IndRNN ini 10 kali lebih cepat dan sedikit lebih akurat dari implementasi *deep learning* Long Short-Term Memory (LSTM) yang sering digunakan. Oleh karena itu, diusulkan pendeteksian kantuk pada penelitian ini menggunakan metode IndRNN.

Diharapkan implementasi *deep learning* ini dapat digunakan untuk membantu pengembang mengembangkan sistem peringatan kantuk yang tentunya membutuhkan pendeteksi kantuk.

1.2 Rumusan Masalah

Dari uraian latar belakang di atas, perumusan masalah untuk penelitian yang akan dilakukan yaitu bagaimana mengimplementasi *deep learning* yang dapat menentukan seseorang itu mengantuk atau tidak berdasarkan fitur-fitur kedipan.

1.3 Batasan Masalah

Agar fokus penelitian tetap terjaga, penelitian ini memiliki beberapa batasan masalah yaitu:

1. Data yang diambil adalah *dataset* The University of Texas at Arlington Real-Life Drowsiness Dataset (UTA-RLDD).
2. Metode yang digunakan adalah *Independently Recurrent Neural Network* (IndRNN).
3. Penelitian ini hanya meneliti pengklasifian 2 macam keadaan mengantuk, yaitu mengantuk dan tidak mengantuk.

1.4 Tujuan Penelitian

Tujuan dari penelitian ini adalah:

1. Mengimplementasikan metode *deep learning* yang dapat menentukan apakah seseorang itu mengantuk atau tidak.
2. Mengetahui konfigurasi model yang baik untuk meningkatkan tingkat keakurasian *deep learning* dalam menentukan apakah seseorang itu mengantuk atau tidak.

3. Mengetahui metode mana yang memiliki akurasi yang lebih tinggi dan *training* yang lebih cepat antara IndRNN dan *Hierarchical Multiscale Long Short-Term Memory* (HM-LSTM).

1.5 Manfaat Penelitian

Manfaat dari penelitian ini adalah untuk mengetahui bahwa IndRNN dapat diterapkan untuk mendeteksi kantuk.

1.6 Sistematika Penulisan

Sistematika penulisan yang digunakan pada tugas akhir ini adalah:

BAB I PENDAHULUAN

Bab ini berisi tentang latar belakang, rumusan masalah, batasan masalah, tujuan penelitian, manfaat penelitian, dan sistematika penulisan.

BAB II TINJAUAN PUSTAKA

Bab ini berisi tentang penelitian-penelitian terdahulu yang berhubungan dengan permasalahan yang diteliti.

BAB III LANDASAN TEORI

Bab ini berisi tentang teori-teori dan konsep yang berhubungan dengan penelitian yang dilakukan dan sebagai pendukung dalam menyelesaikan masalah.

BAB IV METODOLOGI PENELITIAN

Bab ini berisi penjelasan cara kerja pendeteksian kantuk.

BAB V HASIL DAN PEMBAHASAN

Bab ini berisi tentang pembahasan terhadap hasil pengujian.

BAB Vi PENUTUP

Bab ini berisi tentang kesimpulan yang diperoleh dari hasil penelitian yang telah dilakukan serta saran berdasarkan dari hasil penelitian.

BAB II
TINJAUAN PUSTAKA

Penelitian yang telah dilakukan sebelumnya perlu diketahui untuk menghindari adanya duplikasi. Terdapat beberapa penelitian sebelumnya yang menggunakan berbagai macam metode untuk mendeteksi kantuk, diantaranya adalah penelitian oleh Venkata Rami Reddy Chirra et al. (2019) yang penelitiannya bertujuan untuk mendeteksi kantuk berdasarkan keadaan mata dengan *Deep Convolutional Neural Network* (CNN). Cara kerja pendeteksian kantuk pada penelitian tersebut yaitu pertama-pertama mendeteksi wajah pada video lalu ambil bagian mata pada gambar wajah yang terdeteksi. Daerah mata yang diambil tersebut dijadikan sebagai fitur untuk klasifikasi kantuk menggunakan *Deep* CNN. Hasil penelitian menunjukkan bahwa tingkat akurasi metode tersebut sebesar 96,42%.

Ibtissem Belakhdar (2016) mengusulkan menggunakan ANN (*Artificial Neural Network*) untuk mendeteksi kantuk berdasarkan satu saluran *Electroencephalogram* (EEG). ANN tersebut menggunakan fitur yang didapat dari Fast Fourier Transform (FFT) sebagai bahan *training*-nya. Hasil penelitian menunjukkan bahwa tingkat akurasi metode tersebut sebesar 86,5%.

Yingying Jiao et al. (2020) melakukan penelitian pendeteksian kantuk menggunakan GAN (*Generative Adversarial Network*) dan LSTM (*Long Short-Term Memory)* berdasarkan EEG dan *Electrooculography* (EOG). LSTM digunakan untuk membuat model yang dipakai untuk mendeteksi kantuk, sedangkan GAN digunakan untuk menghasilkan *dataset* yang cukup realitis agar meningkatkan performa model. Hasil penelitian menunjukan bahwa tingkat akurasi metode tersebut sebesar 98.14% dengan standar deviasi terkecil sebesar 0,75%.

Esra Vural (2009) melakukan penelitian pendeteksian kantuk berdasarkan 31 macam aktivitas wajah. Aktivitas wajah ini termasuk kedipan dan menguap serta aktivitas wajah lainnya. Aktvitas wajah sebanyak 31 ini digunakan oleh algoritma *machine learning* AdaBoost dan *multinomial ridge regression* sebagai bahan *training*. Hasil penelitian menunjukan bahwa sistem tersebut mampu memprediksi kantuk dengan akurasi 92% untuk sistem menggunakan AdaBoost dan 94% untuk sistem yang menggunakan *multinomial ridge regression.*

Reza Ghoddoosian et al. (2019) melakukan pendeteksian kantuk berdasarkan 4 fitur kedipan yaitu durasi, amplitudo, kecepatan, dan frekuensi. Pendeteksian kantuk ini menggunakan *deep learning* HM-LSTM yang sudah di-*training* dengan *dataset* UTA-RLDD (Ghoddoosian, 2019). Hasil penelitian menunjukkan HM-LTSM mempunyai akurasi 65,2%

dalam pengujian pendeteksian kantuk menggunakan *dataset* UTA-RLDD. Ringkasan penelitian-penelitian sebelumnya dapat dilihat pada Tabel 2.1.

Tabel 2.1 Pebandingan Penelitian Sebelumnya

Judul	Penulis	Tahun	Dataset	Metode	Akurasi
Deep CNN: A Machine Learning Approach for Driver Drowsiness Detection Based on Eye State	Venkata Rami Reddy Chirra, Srinivasulu Reddy Uyyala, dan Venkata Krishna Kishore Kolli	2019	NTHU driver drowsiness	Deep CNN	Memiliki akurasi 96,42%
Detecting driver drowsiness based on single electroencephalography channel	Ibtissem Belakhdar, Walid Kaaniche, Ridha Djmel, dan Bouraoui Ouni	2016	MIT-BIH PolysomnographicW	ANN pada satu saluran EEG	Meiliki akurasi 86,5%
Driver sleepiness detection from EEG and EOG signals using GAN and LSTM networks	Yingying Jiao, Yini Deng, Yun Luo, dan Bao-Liang Lu	2020	Dataset ini dari sukarelawan untuk penelitian ini	GAN dan LSTM	Akurasi GAN dan LSTM adalah 98,14%
Machine Learning Systems for Detecting Driver Drowsiness	Esra Vural, Müjdat Çetin, Aytül Ercil, Gwen Littlewort, Marian Bartlett, dan Javier Movellan	2009	Cohn-Kanade DFAT-504 data set	AdaBoost dan *multinomial ridge regression*	Akurasi AdaBoost adalah 92% dan *multinomial ridge regression* adalah 94%
A Realistic Dataset and Baseline Temporal Model for Early Drowsiness Detection	Reza Ghoddoosian, Marnim Galib, dan Vassilis Athitsos	2019	UTA-RLDD	HM-LSTM	

BAB III
LANDASAN TEORI

3.1 Kantuk

Kantuk atau ngantuk adalah kondisi ketika seseorang merasa ingin tidur. Kondisi ini biasa terjadi pada malam hari ataupun terkadang di siang hari yang merupakan hal yang wajar. Namun jika rasa kantuk terjadi secara berlebihan hingga mengganggu aktivitas dan menurunkan produktivitas, maka kondisi ini perlu ditangani.

Rasa kantuk umumnya muncul karena kurang tidur. Meskipun terlihat sederhana, kantuk bisa memicu timbulnya berbagai masalah, misalnya mengganggu prestasi di sekolah, atau produktivitas di kantor, memengaruhi emosi, dan menyebabkan kecelakaan, baik di jalan raya maupun di lingkungan kerja.

Kantuk memang merupakan hal yang wajar, namun jika terjadi sesuatu secara tidak normal, bisa saja rasa ngantuk tersebut merupakan tanda dari suatu penyakit, seperti *sleep apnea*, narkolepsi, insomnia, depresi, gangguan kecemasan, atau diabetes (Dr.Tjin Willy, 2019).

3.2 Pendeteksian Wajah Dengan HOG (Histogram of Oriented Gradients)

HOG ini merupakan salah satu teknik untuk mengenali objek manusia terutama pada bagian wajah pada suatu citra. Metode ini mengandalkan histogram lokal dari orientasi gradien pada gambar yang telah dinormalisasi pada petak yang padat (Dalal, 2005). Input HOG ini berupa representasi piksel seperti RGB dan *grayscale*. RGB adalah ruang warna atau kumpulan warna yang mungkin dari perpaduan 3 warna, 3 warna tersebut adalah merah (R), hijau (G), biru (B). Berdasarkan langkahnya, proses awal pada metode HOG adalah sebagi berikut:

3.2.1 Normalisasi

Normalisasi berfungsi untuk menyeragamkan kecerahan pada suatu gambar, yang nantinya gambar tersebut menjadi lebih mudah digunakan oleh *machine learning*. Normalisasi dilakukan dengan mengaplikasikan operasi *gamma correction* untuk mengubah kecerahan pada setiap piksel. *Gamma correction* adalah operasi non-linear untuk *encode* dan *decode* kecerahan. *Gamma correction* ini biasanya berupa operasi yang menggunakan persamaan *power law* seperti pada persamaan 3.1.

$$V_{out} = AV_{in}^{\gamma} \quad (3.1)$$

V_{in} adalah nilai piksel yang ingin diubah kecerahannya. V_{in} berpangkat *gamma value* γ dan dikali dengan bilangan konstan A akan menghasilkan V_{out} yaitu nilai piksel yang sudah diubah kecerahannya. Semakin besar γ maka gambar semakin cerah, semakin kecil γ gambar semakin redup.

3.2.2 Hitung Besaran Gradien dan Orientasi Gradien

Setelah normalisasi dilakukan, besaran gradien dan orientasi gradien pada setiap piksel dihitung menggunakan *mask* yang dipilih. Persamaan besaran gradien dan orientasi gradien dapat dilihat pada persamaan 3.2 dan 3.3.

$$\text{Besaran Gradien} = \sqrt{s_x^2 + s_y^2} \quad (3.2)$$

$$\text{Orientasi Gradien} = \tan^{-1}\left(\frac{S_y}{S_x}\right) \quad (3.3)$$

S_x dan S_y secara urut adalah nilai gradien dari *mask* arah horizontal dan arah vertikal.

3.2.3 Orientation Binning

Orientation binning adalah mengumpulkan besaran gradien berdasarkan orientasi gradien. Sebelum *orientation binning* dilakukan, gambar dibagi menjadi beberapa *cells*. Pada tiap *cells* inilah, *Orientation Binning* dilakukan. Hasil *orientation binning* ini berupa *histogram of oriented gradient,* yaitu histogram yang menunjukkan besaran gradien untuk tiap orientasi gradien.

3.2.4 Klasifikasi dengan SVM

Histogram of oriented gradient yang didapatkan diubah menjadi array 1 dimensi agar dapat diklasifikasi oleh Linear SVM untuk menentukan apakah histogram gradien itu adalah wajah atau bukan.

3.3 Pendeteksian Facial Landmark dengan Ensemble of Regression Trees

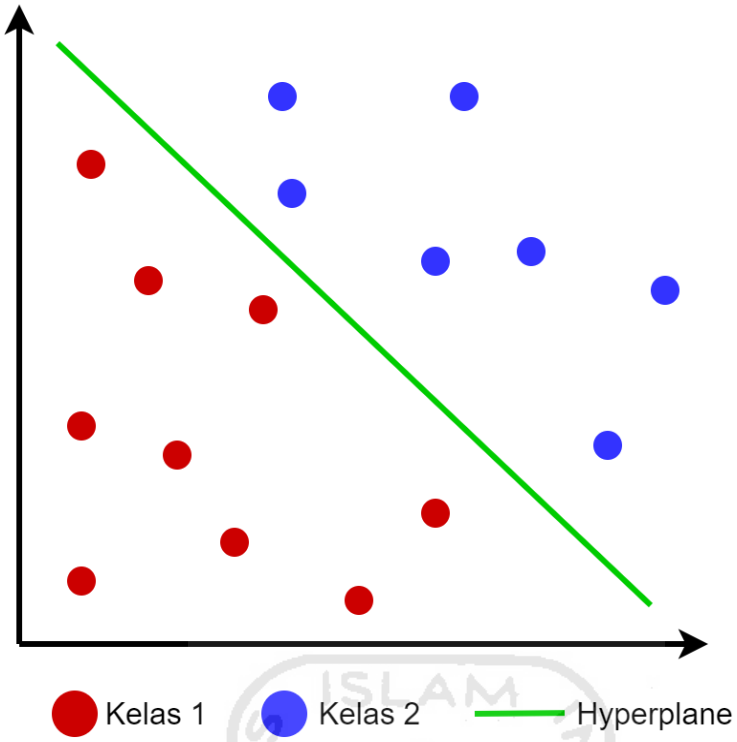
Pendeteksian ini memanfaatkan *Cascade of Regressors* untuk memprediksi *facial landmark*. Metode pendeteksian ini diusulkan oleh Kazemi dan Sullivan (2014). Misal $x_i \in \mathbb{R}^N$ adalah koordinat x dan y pada *i*th *facial landmark* pada gambar *I*. Vektor $S = (x_1^T, x_2^T, \ldots, x_p^T)^T \in \mathbb{R}^{2p}$ menyatakan semua koordinat *p facial landmark* pada *I*, sehingga vektor S bisa disebut sebagai *shape*. $\hat{S}^{(t)}$ menyatakan perkiraan S pada suatu iterasi yang sekarang dilakukan. Setiap *regressor*, $r_t(.\ ,\ .)$, di tiap iterasi memprediksi sebuah perbaruan vektor dari gambar dan $\hat{S}^{(t)}$ yang ditambah dengan *shape* yang sekarang untuk meningkatkan akurasi perkiraan:

$$\hat{S}^{(t+1)} = \hat{S}^{(t)} + r_t(I, \hat{S}^{(t)}) \qquad (3.4)$$

Regressor r_t adalah yang bertugas memprediksi *facial landmark* berdasarkan fitur seperti intensitas piksel yang merupakan hasil komputasi dari *I* dan indeks relatif terhadap perkiraan *shape* sekarang $\hat{S}^{(t)}$.

3.4 Support Vector Mahinces (SVM)

SVM adalah pengklasifikasi yang menggunakan prinsip *Structural Risk Minimization* (SRM) untuk menemukan *hyperplane* terbaik yang memisahkan dua kelas (Nugroho et al., 2003). SRM adalah prinsip induktif yang digunakan dalam *machine learning.* SVM ini merupakan pengembangan dari kombinasi beberapa konsep sebelumnya, yaitu *margin hyperplane* dan *kernel*. Prinsip dasar SVM ini adalah klasifikasi linear yang kemudian dikembangkan agar dapat mengatasi masalah *non-linear*. Gambar 3.1 adalah ilustrasi bagaimana SVM memisahkan dua kelas dengan *hyperplane*.

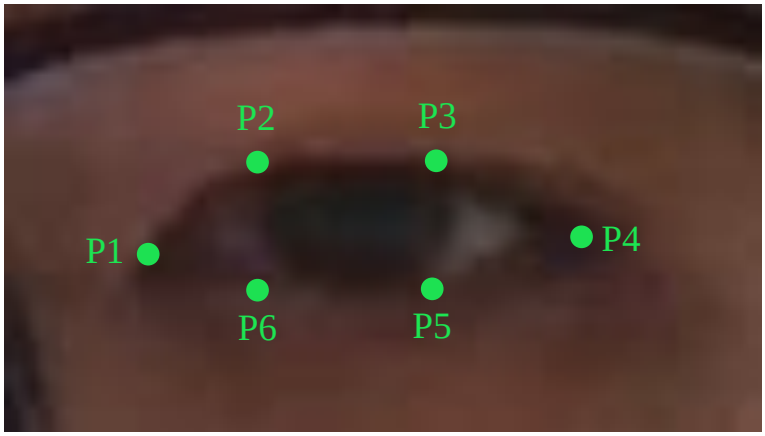


Gambar 3.1 Ilustrasi bagaimana SVM memisahkan dua kelas dengan *hyperplane*

3.5 Pendeteksian Kedipan dengan Eye Aspect Ratio (EAR)

Pendeteksian ini memanfaatkan *facial landmark* pada daerah mata dan SVM untuk klasifikasi. Pendeteksian ini diusulkan oleh Soukupová dan Čech (2016). *Facial landmark* yang didapatkan akan digunakan untuk menghitung *Eye Aspect Ratio* (EAR) yaitu ukuran seberapa terbuka suatu mata. EAR ini dihitung berdasarkan persamaan 3.5 yaitu p merupakan koordinat *facial landmark*. Koordinat p ini adalah koordinat pada mata sesuai dengan Gambar 3.2. Serangkaian EAR inilah yang digunakan sebagai fitur dalam klasifikasi kedipan dengan SVM.

$$EAR = \frac{\|\overrightarrow{p_2} - \overrightarrow{p_6}\| + \|\overrightarrow{p_3} - \overrightarrow{p_5}\|}{2\|\overrightarrow{p_1} - \overrightarrow{p_4}\|} \qquad (3.5)$$



Gambar 3.2 Koordinat *facial landmark* pada mata

3.6 Algoritma Pengambilan Kedipan

Ghoddoosian et al. (2019) menemukan pendeteksian kedipan yang telah dikembangkan oleh Soukupová dan Jan Čech (2016) masih belum sempurna, yang mana terdapat kesalahan berupa dua kali kedipan dianggap hanya satu kali kedipan. Oleh karena itu Ghoddoosian et al. membuat algoritma pengambilan kedipan untuk menyempurnakan kesalahan tersebut. Cara kerja algoritma ini adalah sebagai berikut:

Definisikan deretan EAR $x \in \mathbb{R}^M$, M adalah ukuran time series x, sebagai kandidat satu atau lebih kedipan dan epsilon=0.01. Langkah-langkah algoritma ini yaitu:

1. $\dot{x}[n] \leftarrow x[n+1] - x[n], \forall\, n \in \{i | i = 0,1, \ldots, M-2\}$
2. if $\dot{x}[0] = 0$ then $\dot{x}[0] \leftarrow -1 \times$ epsilon
3. $\dot{x}[n] \leftarrow \dot{x}\,[n-1] \times$ epsilon, $\forall\, n \in \{i | \dot{x}[i] = 0 \wedge i \neq 0\}$
4. $c[n] \leftarrow \dot{x}[n+1] \times \dot{x}[n], \forall\, n \in \{\, i | i = 0,1, \ldots, M-3\}$
5. $e \in \mathbb{R}^{P+2}$, $P \leq M-2$ untuk menyimpan indeks extrem P
6. $e[0] \leftarrow 0$, $e[P+1] \leftarrow M-1$
7. $e[k] \leftarrow n+1, \forall\, (n \in \{i | c[i] < 0\} \wedge k \in \{i | I = 1, 2, \ldots, P\})$
8. $THR \leftarrow 0.6 \times \max(x) + 0.4 \times \min(x)$, sebagai ambang batas
9. $t \in \mathbb{R}^{P+2}$, untuk menyimpan +1 (di atas ambang batas) atau -1 (di bawah ambang batas)
10. $t[0] \leftarrow +1$, $t[P+1] \leftarrow +1$
11. Tambah +1 ke dalam t untuk setiap $n \in \{i | x[e[i]] > THR\}$ dan tambah -1 ke dalam t untuk setiap $n \in \{i | x[e[i]] \leq THR\}$, berurutan sesuai dengan indeks pada e
12. $z \in \mathbb{R}^{P+1}$, $z[n] \leftarrow t[n+1] \times t[n]$
13. s, untuk menyimpan semua indeks yang berisi nilai negatif di dalam z yang mempresentasikan arah bawah dan atas suatu pergerakan mata dalam kedipan

14. N ← $\frac{\text{length(s)}}{2}$, N adalah banyak kedipan baru
15. for i ← 0 to N – 1 do
16. StartIndex ← e[s[2 × i]]
17. EndIndex ← e[s[2 × i + 1] + 1]
18. BottomIndex ← e[s[2 × i + 1]]

Hasil algoritma ini berupa titik *StartIndex*, *EndIndex*, dan *BottomIndex* pada N kedipan yang didapatkan dari x. *StartIndex* adalah nomor *frame* saat mata terbuka di awal siklus kedipan. *EndIndex* adalah nomor *frame* saat mata terbuka di akhir siklus kedipan. *BottomIndex* adalah nomor *frame* saat mata tertup dalam siklus kedipan.

3.7 Penghitungan Nilai 4 Fitur Kedipan

Rangkaian nilai EAR yang terdeteksi sebagai kedipan akan dimasukkan ke dalam persamaan yang digunakan untuk mendapatkan nilai 4 fitur kedipan, yaitu persamaan 3.6 untuk mendapatkan durasi, persamaan 3.7 untuk mendapatkan amplitudo, persamaan 3.8 untuk mendapatkan kecepatan mata membuka, dan persamaan 3.9 untuk mendapatkan frekuensi.

$$Durasi_i = end_i - start_i + 1 \tag{3.6}$$

$$Amplitudo_i = \frac{EAR[start_i] - 2EAR[bottom_i] + EAR[end_i]}{2} \tag{3.7}$$

$$Kecepatan\ Mata\ Membuka_i = \frac{EAR[end_i] - EAR[bottom_i]}{end_i - bottom_i} \tag{3.8}$$

$$Frekuensi_i = 100 \times \frac{Number\ of\ blinks\ up\ to\ blink_i}{Number\ of\ frames\ up\ to\ end_i} \tag{3.9}$$

$start_i$, $bottom_i$, dan end_i secara urut merupakan *frame* saat mata dalam keadaan terbuka, *frame* saat mata dalam keadaan tertutup, dan *frame* saat mata terbuka setelah mata tertutup dalam satu siklus kedipan.

3.8 Deep Learning

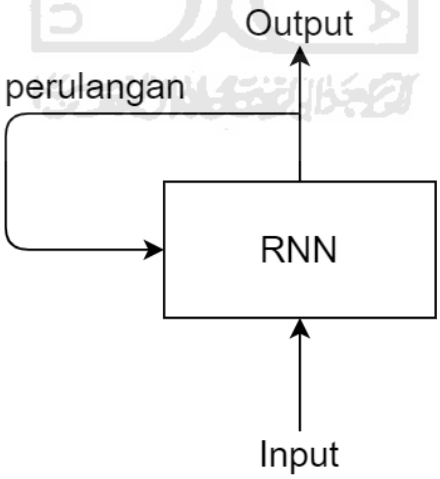
Deep learning merupakan bagian dari bidang *machine learning* yang menekankan pembelajaran representasi dari data dengan *layer* yang semakin dalam *layer*-nya maka semakin

berarti representasi data yang dipelajari. *Deep learning* ini biasanya memiliki banyak *layer,* karena mempunyai *layer* yang banyak inilah yang membedakan dengan metode *machine learning* pada umumnya yang hanya memiliki satu atau dua *layer*. Pembelajaran *deep learning* biasanya dilakukan melalui model yang bernama *neural networks* yang merupakan pembelajaran yang terinspirasi dari cara kerja otak. Oleh karena itu, biasanya *deep learning* mempunyai neuron. Neuron dalam *deep learning* adalah fungsi matematika yang digunakan untuk pemrosesan data.

3.9 Recurrent Neural Neworks (RNN)

RNN merupakan salah satu metode *deep learnig* yang dapat mempelajari data temporal atau rangkaian nilai data yang berurutan (Chollet, 2017). Berbeda dengan Convolution Neural Network (RNN), RNN memiliki memori yang dapat digunakan untuk menyimpan *state*. *State* yang dimaksud adalah informasi yang dapat digunakan untuk merepresentasikan data temporal. Contoh data temporal adalah data nilai mata uang per hari dari tanggal 1 September sampai 30 september dan kata-kata pada suatu kalimat.

Cara kerja RNN yaitu memproses data temporal dengan melakukan perulangan untuk setiap elemen data temporal dan mengelola *state* yang berisi informasi hasil dari pemrosesan elemen data temporal. Oleh karena itu, arsiktertur RNN diilustrasikan dengan proses perulangan seperti pada Gambar 3.3.



Gambar 3.3 Arsitektur RNN

3.10 Hierarchical Multiscale Long Short-Term Memory (HM-LSTM)

HM-LSTM merupakan RNN yang memanfaatkan pendeteksi batas berparameter, yang mempelajari kapan suatu segmen berhenti agar mendapatkan hasil yang optimal (Chung et al., 2017) . Pendeteksi batas ini digunakan pada setiap perubahan waktu pada tiap *layer* untuk

menentukan operasi apa yang akan dipakai dari 3 operasi yang ada, operasi tersebut adalah UPDATE, COPY, FLUSH. Operasi UPDATE adalah operasi yang memperbarui nilai variabel berdasarkan hasil penghitungan yang dilakukan, COPY adalah operasi yang memperbarui suatu nilai variabel dengan nilai dari variabel lain, FLUSH adalah operasi yang mengembalikan nilai variabel pada keadaan awal. Cara pemilihan operasi tersebut yaitu diplih berdasarkan nilai keadaan pendeteksi batas pada waktu sekarang di *layer* bawahnya, yang bersimbol z_t^{l-1} dan pendeteksi batas pada waktu sebelumnya di *layer* sekarang berada, yang bersimbol z_{t-1}^{l}. Operasi yang dipilih tersebut digunakan untuk menghitung nilai *cell state* pada waktu sekarang di *layer* yang sekarang berada. *Cell state* ini nantinya digunakan untuk menghitung *hidden state*. *Hidden state* ini adalah representasi sekuens, *hidden state* inilah yang digunakan untuk mengklasifikasi sekuens.

Cell state dihitung berdasarkan nilai *forget gate*, *input gate*, dan kandidat *cell state*. *Forget gate* menentukan apa yang dibuang pada *hidden state*. *Input gate* menentukan informasi apa yang disimpan. Kandidat *cell state* adalah nilai yang mungkin ditambahkan ke *cell state*. Persamaan 3.10 menunjukkan penghitungan *cell state* dengan menggunakan operasi yang ditentukan oleh pendeteksi batas z. *Cell state* dinyatakan sebagai variabel c, *forget gate* dinyatkan sebagai variabel f, *input gate* dinyatakan sebagai variabel i, kandidat *cell state* dinyatakan sebagai g, simbol l adalah nomor *layer*, dan simbol t adalah nomor waktu pada suatu sekuens.

$$\mathrm{c}_t^l = \begin{cases} \mathrm{f}_t^l \odot \mathrm{c}_{t-1}^l + \mathrm{i}_t^l \odot \mathrm{g}_t^l & \text{if } \mathrm{z}_{t-1}^l = 0 \text{ dan } \mathrm{z}_t^{l-1} = 1 \text{ (UPDATE)} \\ \mathrm{c}_{t-1}^l & \text{if } \mathrm{z}_{t-1}^l = 0 \text{ dan } \mathrm{z}_t^{l-1} = 0 \text{ (COPY)} \\ \mathrm{i}_t^l \odot \mathrm{g}_t^l & \text{if } \mathrm{z}_{t-1}^l = 1 \text{ (FLUSH)} \end{cases} \tag{3.10}$$

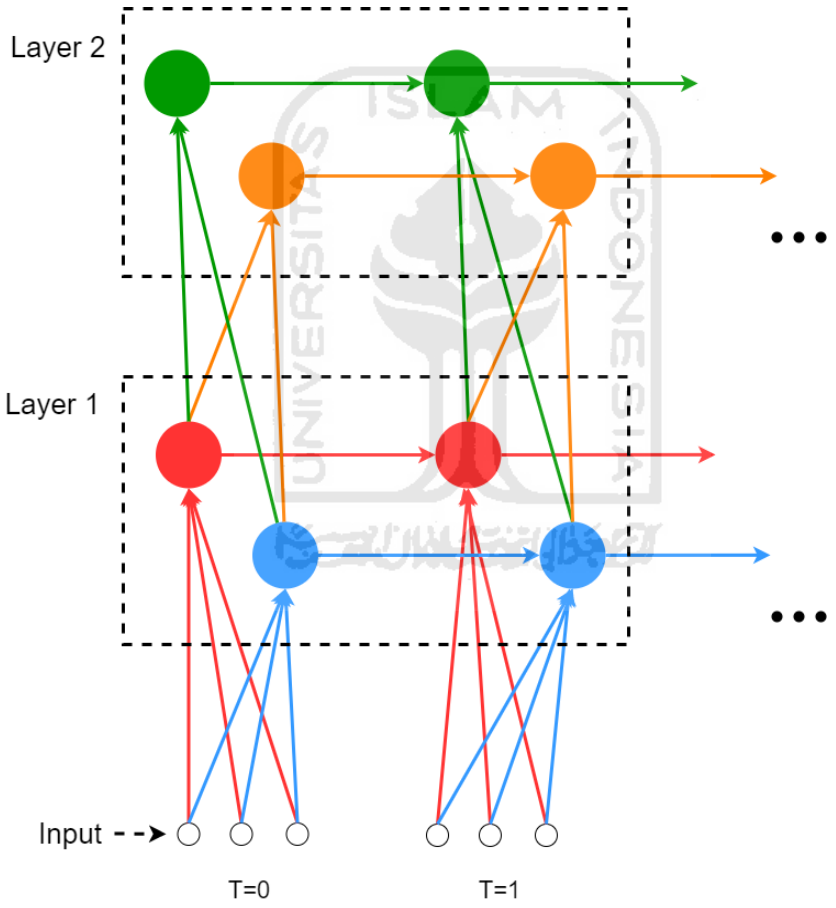
Persamaan 3.11 menunjukkan penghitungan *hidden state* dan operasinya pada suatu kondisi. Variabel o merupakan *output gate*. *Output gate* menentukan bagian mana pada *cell state* sekarang yang akan dipakai pada waktu berikutnya.

$$\mathrm{h}_t^l = \begin{cases} \mathrm{h}_{t-1}^l & \text{if COPY} \\ \mathrm{o}_t^l \odot \tanh\left(\mathrm{c}_t^l\right) & \text{otherwise} \end{cases} \tag{3.11}$$

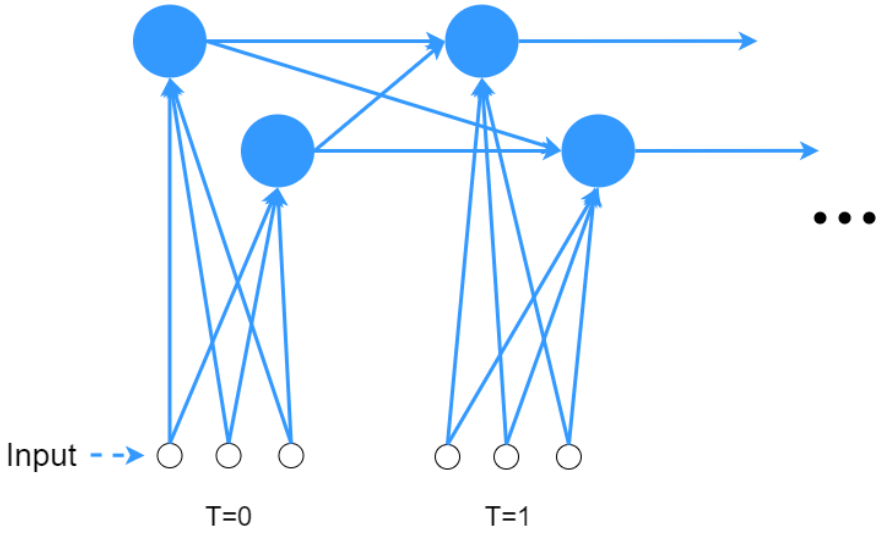
3.11 Independently Recurrent Neural Networks (IndRNN)

IndRNN adalah *deep learning* hasil dari pengembangan RNN yang dapat mengenali pola temporal. IndRNN ini diusulkan oleh Shuali Li et al. (2019). Perbedaan IndRNN dan RNN ini

dapat dilihat pada pemrosesan neuron yaitu pada IndRNN, pemrosesan setiap neuron pada suatu *layer* tidak bergantung dengan *layer* lain sebagaimana diilustrasikan pada Gambar 3.4, sedangkan pada RNN, setiap neuron terhubung satu sama lain dalam satu *layer*, sebagaimana diilustrasikan pada Gambar 3.5. Pada Gambar 3.4 dan 3.5, lingkaran besar adalah neuron, lingkaran kecil putih adalah input, tanda panah menunjukkan hubungan antar neuron, dan deretan 3 buat lingkaran hitam kecil menunjukkan waktu atau T seterusnya. Pemrosesan neuron yang independen ini memungkinkan IndRNN mendapatkan representasi sekuens yang dapat menangkap pola temporal yang lebih banyak dari RNN biasa. Setiap neuron terdapat proses penghitungan *hidden state*, *hidden state* disini yang dimaksud adalah representasi sekuens. Cara menghitung *hidden state* pada setiap neuron dapat lihat pada persamaan 3.12.



Gambar 3.4 Hubungan Antar Neuron pada IndRNN



Gambar 3.5 Hubungan antar neuron pada RNN biasa

$$h_t = \sigma(Wx_t + u \odot h_{t-1} + b) \tag{3.12}$$

Penjelasan persamaan 3.12 yaitu $x_t \in \mathbb{R}^M$ dan $h_t \in \mathbb{R}^N$ adalah input dan *hidden state* pada langkah waktu *t*. $W \in \mathbb{R}^{NxM}$, $u \in \mathbb{R}^N$ dan $b \in \mathbb{R}^N$ secara urut adalah bobot input yang sekarang dan input yang berulang (*recurrent*) serta bias neuron. $\odot$ adalah *hadamard product*, σ fungsi aktivasi neuron *element-wise* dan N adalah banyaknya neuron di dalam *layer* RNN. Dibandingkan dengan RNN tradisional, bobot *recurrent* u di dalam IndRNN merupakan vektor dan proses input *recurrent* dengan perkalian vektor *element-wise*. Setiap neuron di setiap *layer* tidak bergantung terhadap neuron yang lain, karena itulah *recurrent* pada IndRNN ini disebut dengan *indepedently recurrent*.

BAB IV
METODOLOGI PENELITIAN

4.1 Pengumpulan Data

Data yang digunakan pada penelitian ini dapat dikategorikan sebagai data sekunder. Data sekunder adalah data yang tidak diperoleh langsung dari peneliti. Data ini diambil dari UTA-RLDD (Ghoddoosian, 2019). *Dataset* ini berisi video peserta saat sedang tidak mengantuk, agak mengantuk, dan mengatuk. Berikut adalah keterangan *dataset* UTA-RLDD:

a. Total peserta yang menyumbang video adalah 60 orang.
b. Peserta tersebut terdiri dari berbagai macam etnis, yaitu 10 Kaukasian (orang kulit putih), 5 Hispanik bukan kulit putih, 30 Indo-Arya dan Dravida, 8 Timur Tengah, 7 Asia Timur.
c. Sebanyak 7 peserta yang berkacamata.
d. Jumlah video adalah 180 buah.
e. Setiap peserta menyumbang 3 video yang terdiri dari video saat tidak mengantuk, video saat agak mengantuk, dan video saat mengantuk.
f. Rentang umus peserta adalah 20 sampai 59 tahun.
g. Video pada *dataset* ini direkam dengan menggunakan *smartphone* atau *webcam* dengan frame rate 28-30 fps.
h. Resolusi yang ada pada dataset ini adalah 720p dan 1080p

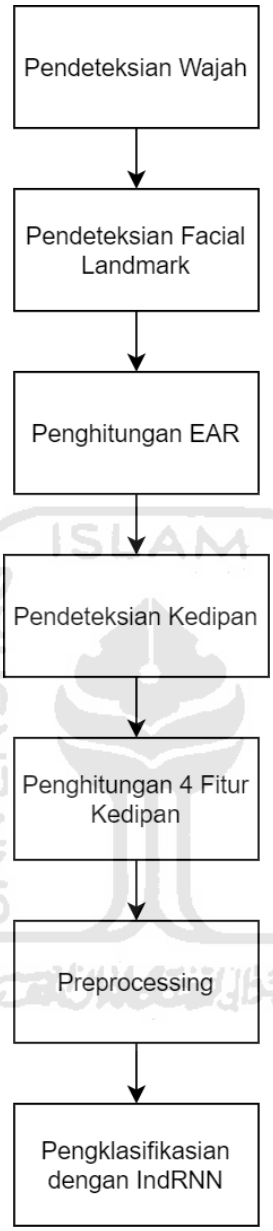
Pada penelitian ini, data yang diambil adalah hanya video orang mengantuk dan tidak mengantuk yang totalnya berjumlah 120 video. Dari 120 video tersebut, 96 video digunakan untuk *training*, 12 video untuk validasi dan sisanya 12 video untuk pengujian. Sampel gambar dari video dari 3 macam keadaan mengantuk dapat dilihat pada Gambar 4.1.



Gambar 4.1 Sampel gambar orang tidak mengantuk (baris pertama), orang agak mengantuk (baris kedua), dan orang mengantuk (baris ketiga)
(Sumber: sites.google.com/view/utarldd/home)

4.2 Alur Pendeteksian Kantuk

Langkah-langkah pendeteksian alur kantuk sesuai dengan *flowchart* pada Gambar 4.2.



Gambar 4.2 *Flowchart* alur pendeteksian kantuk

Alur pendeteksian kantuk dimulai dengan pendeteksian wajah, pendeteksian *facial* landmark, penghitungan EAR, pendeteksian kedipan, penghitugan 4 fitur kedipan, normalisasi, dan yang terakhir adalah pengklasifikasian dengan IndRNN. Untuk penjelasan tahap-tahap tersebut adalah sebagai berikut:

4.2.1 Pendeteksian Wajah

Alur kerja pendeteksian wajah adalah sebagai berikut:

a. Normalisasi

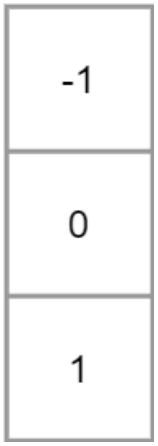
Model warna yang digunakan untuk mendeteksi wajah adalah *grayscale* yaitu nilai piksel dengan rentang nilai dari 0 sampai 255 sehingga persamaan *gamma correction* dapat dilihat pada persamaan 4.1.

$$V_{out} = (\frac{1}{V_{in}})^{\frac{1}{\gamma}} \quad (4.1)$$

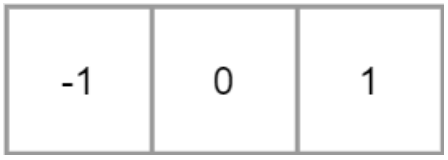
V_{in} adalah nilai piksel yang ingin diubah kecerahannya. V_{out} yaitu nilai piksel yang sudah diubah kecerahannya. γ adalah *gamma value*.

b. Hitung Besaran Gradien dan Orientasi Gradien

Sebelum menghitung besaran gradien dan orientasi gradien untuk setiap piksel, kita harus memilih *mask* yang dipakai. Berdasarkan penelitian oleh Dalal (2005), *centered mask* merupakan *mask* yang mempunyai performa yang paling bagus. *Centered mask* dibagi menjadi 2 yaitu arah vertikal sesuai dengan Gambar 4.3 dan arah horizontal sesuai dengan Gambar 4.4. Angka-angka pada Gambar 4.3 dan 4.4 merupakan koefisien *mask* yang digunakan untuk menghitung besaran gradien dan orientasi gradien.

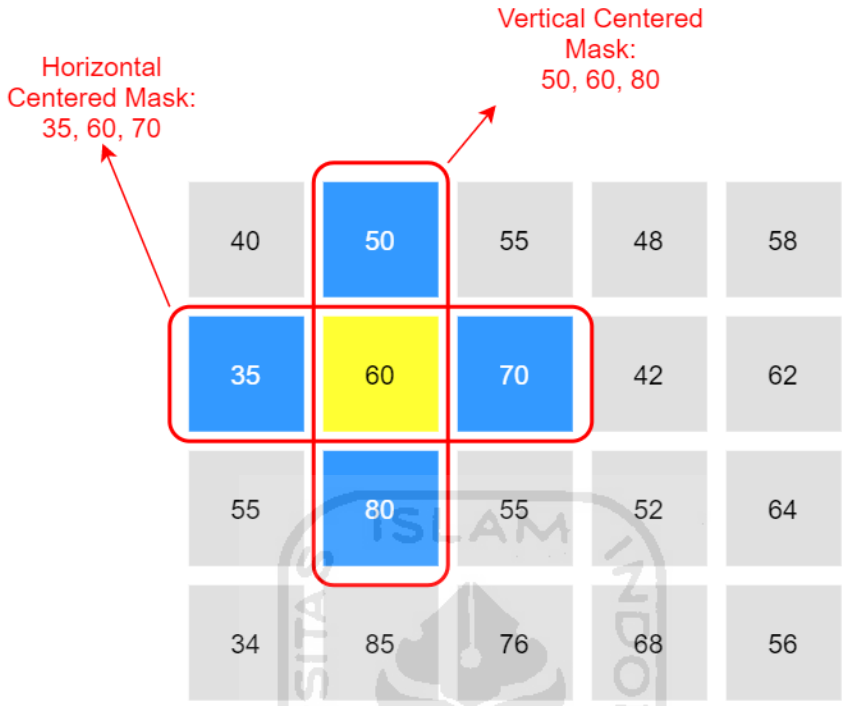


Gambar 4.3 *Centered mask* arah vertikal



Gambar 4.4 *Centered mask* arah horizontal

Berikut adalah contoh penghitungan besaran gradien dan orientasi gradien dengan *centered mask* pada Gambar 4.5 yang tepatnya ditujukan untuk nilai piksel yang diwarnai kuning pada Gambar 4.5:



Gambar 4.5 Contoh nilai piksel pada suatu gambar berukuran 5x4 piksel

Misal:

Mask Coefficient $= [-1, 0, 1]$

Pixel on Horizontal Mask = [35, 60, 70]

Pixel on Vertical Mask = [50, 60, 80]

Maka:

$S_x =$ Pixel on Horizontal Mask $\cdot$ Mask Coefficient

$S_x = (35 \times -1) + (60 \times 0) + (70 \times 1)$

$S_x = -35 + 70 = 35$

$S_y =$ Pixel on Vertical Mask $\cdot$ Mask Coefficient

$S_y = (50 \times -1) + (60 \times 0) + (80 \times 1)$

$S_y = -50 + 80 = 30$

$$\text{Besaran Gradien} = \sqrt{s_x^2 + s_y^2}$$

$$\text{Besaran Gradien} = \sqrt{(35)^2 + (35)^2}$$

$$\text{Orientasi Gradien} = \tan^{-1}\left(\frac{s_y}{s_x}\right)$$

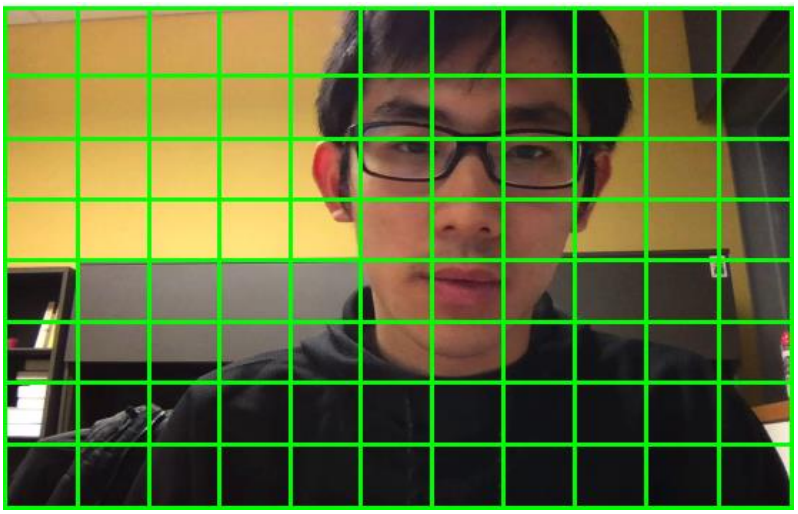
$$\text{Orientasi Gradien} = \tan^{-1}\left(\frac{30}{35}\right)$$

$$\text{Orientasi Gradien} = 40.6$$

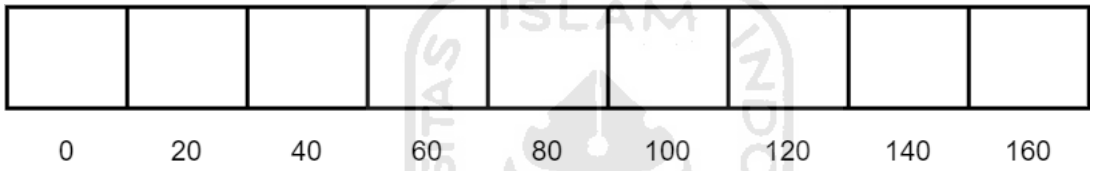
S_x didapatkan dengan mengaplikasikan *Centered Mask* arah horizontal pada piksel yang diwarnai kuning yang piksel tersebut biasa disebut sebagai piksel pusat. S_y didapatkan dengan mengaplikasikan *Centered Mask* arah vertikal pada piksel pusat.

c. Orientation Binning

Setelah setiap piksel dihitung besaran gradien dan orientasi gradiennya, piksel dikelompokkan menjadi beberapa *cell* seperti ilustrasi pada Gambar 4.6 yang sudah ditandai dengan garis hijau. Setelah dikelompokkan, barulah *orientation binning* dilakukan. Dalam proses *Orientation binning* terdapat *orientation bin* yang merupakan tempat dikumpulkannya besaran gradien berdasarkan orientasi gradien. Berdasarakan penelitian oleh Dalal (2005), penggunaan 9 *orientation bin* yang didapatkan dari pembagian orientasi 0-180° mempuyai akurasi yang terbaik. Pembagian orientasi 0-180° pada *orientation bin* ini dibagi sama rata sesuai dengan ilustrasi pada Gambar 4.7. *Orientation bin* yang berjumlah 9 dengan pembagian orientasi dari 0-180° ini dipilih berdasarkan penelitian yang dilakukan oleh Dalal (2005) yang menunjukkan pembagian *Orientation bin* dengan 9 orientasi dari 0-180° memiliki performa yang paling bagus.



Gambar 4.6 Gambar wajah yang sudah dibagi menjadi beberapa *cell*



Gambar 4.7 Bin sebanyak 9 sebuah yang dinamai berdasarkan orientasi dari 0-180°

Proses o*rientation binning* dilakukan berdasarkan persamaan 4.5 dan 4.6.

$$Ml = \frac{(d - (r - bl))}{d} \times m \quad (4.5)$$

$$Mh = \frac{(d - (bh - r))}{d} \times m \quad (4.6)$$

Variabel m adalah nilai besaran gradien suatu piksel, variabel r adalah orientasi gradien suatu piksel, dan Variabel d adalah jarak orientasi antar *orientation bin* pada suatu *cell*. bl adalah orientasi suatu *orientation bin* yang orientasi pada *orientation bin* tersebut kurang dari sama dengan r dan memiliki jarak antara bl dan r terkecil dari semua *orientation bin* yang ada pada *cell* tersebut. Ml adalah nilai besaran gradien yang akan dimasukkan ke dalam *orientation bin* berorientasi bl. bh adalah orientasi suatu *orientation bin* yang orientasi pada *orientation bin* tersebut lebih dari r, memiliki jarak antara bh dan r terkecil dari semua *orientation bin* yang ada pada suatu *cell*, dan jika tidak *orientation bin* yang lebih besar dari nilai r, maka nilai

bh adalah nilai orientasi tertinggi dari semua *orientation bin* ditambah dengan nilai d. Mh adalah nilai besaran gradien yang akan dimasukkan ke dalam *orientation bin* berorientasi bh.

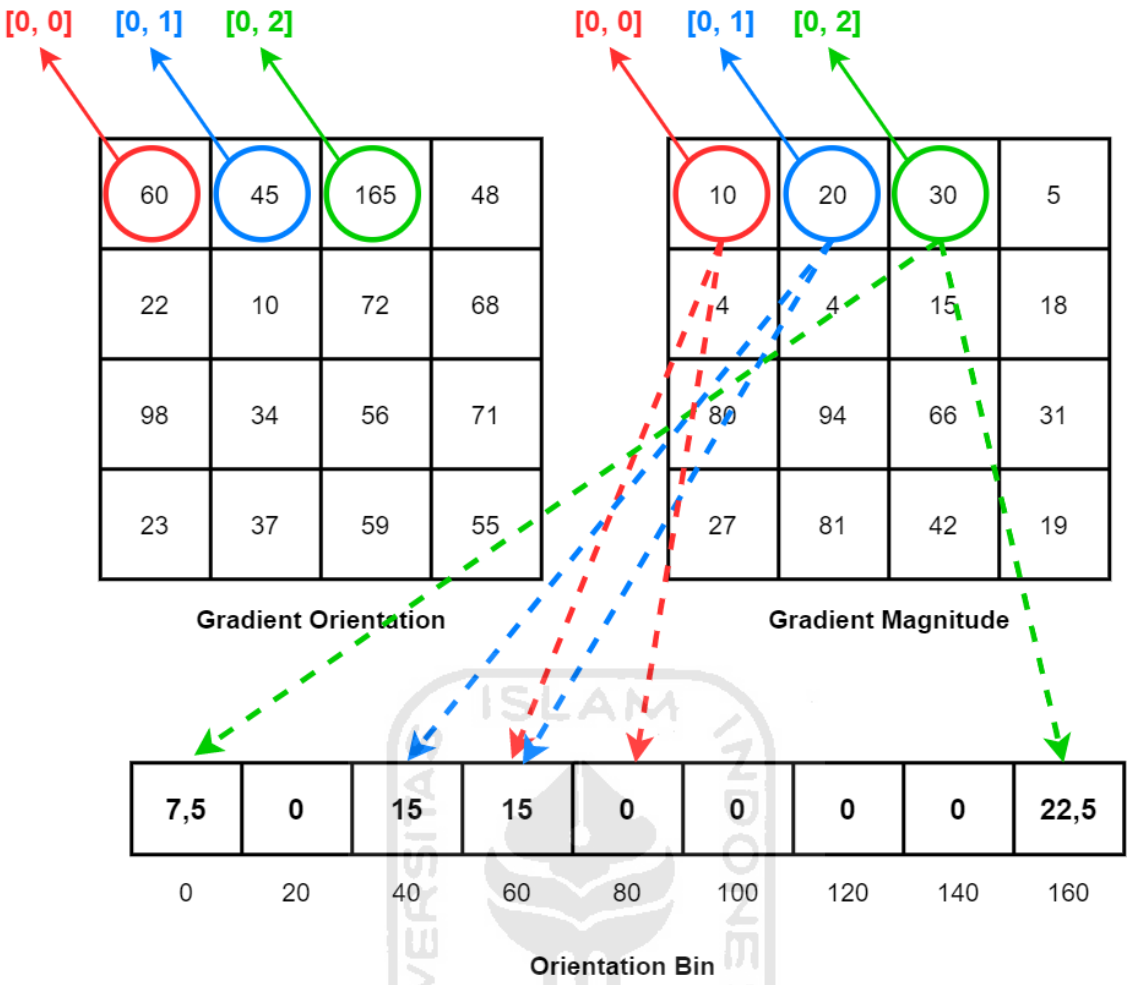
Misal tiap *cell* pada Gambar 4.6 tersebut terdapat 4 x 4 piksel, maka contoh proses *orientation binning* pada satu *cell* yang besaran gradien dan orientsi gradiennya sesuai dengan Gambar 4.8 dan 4.9 adalah diilustrasikan pada Gambar 4.10.

60	45	165	48
22	10	72	68
98	34	56	171
123	107	159	105

Gambar 4.8 Besaran gradien tiap piksel

10	20	30	5
4	4	15	18
80	94	66	31
27	81	42	19

Gambar 4.9 Orientasi gradien tiap piksel



Gambar 4.10 Contoh proses *orientation binning*

Gambar 4.10 menunjukkan proses *Orientation Binning* untuk piksel [0, 0], [0, 1], dan [0, 2]. Penjelasan prosesnya adalah sebagai berikut:

1. Piksel [0, 0] mempunyai variabel r yang bernilai 60 sehingga nilai bl adalah 60, karena orientasi 60 kurang dari sama dengan r dan merupakan *orientasi bin* terdekat sedangkan nilai bh adalah 80, karena orientasi 80 lebih dari r dan merupakan *orientasi bin* terdekat. Proses *orientation binning* pada piksel [0, 0] adalah sebagai berikut:

   Proses penghitungan besaran gradien Ml yang akan dimasukkan ke dalam *orientation bin* 60:

$$Ml = \frac{(d - (r - bl))}{d} \times m$$

$$Ml = \frac{(20 - (60 - 60))}{20} \times 10$$

$$Ml = \frac{20}{20} \times 10$$
$$Ml = 10$$

Nilai *orientation bin* sekarang:

0	0	0	10	0	0	0	0	0
0	20	40	60	80	100	120	140	160

Proses penghitungan besaran gradien Mh yang akan dimasukkan ke dalam *orientation bin* 80:

$$Mh = \frac{(d - (bh - r))}{d} \times m$$
$$Mh = \frac{(20 - (80 - 60))}{20} \times 10$$
$$Mh = \frac{0}{20} \times 20$$
$$Mh = 0$$

Nilai *orientation bin* sekarang:

0	0	0	10	0	0	0	0	0
0	20	40	60	80	100	120	140	160

2. Untuk piksel [0, 1] dengan r bernilai 45, hasil besaran gradiennya dikumpulkan pada *orientation bin* 40 dan 60:

Proses penghitungan besaran gradien Ml yang akan dimasukkan ke dalam *orientation bin* 60:

$$Ml = \frac{(d - (r - bl))}{d} \times m$$
$$Ml = \frac{(20 - (45 - 40))}{20} \times 20$$
$$Ml = \frac{15}{20} \times 20$$
$$Ml = 15$$

Nilai *orientation bin* sekarang:

0	0	15	10	0	0	0	0	0
0	20	40	60	80	100	120	140	160

Proses penghitungan besaran gradien Mh yang akan dimasukkan ke dalam *orientation bin* 60:

$$\text{Mh} = \frac{(\text{d} - (\text{bh} - \text{r}))}{\text{d}} \times \text{m}$$
$$\text{Mh} = \frac{(20 - (60 - 45))}{20} \times 20$$
$$\text{Mh} = \frac{5}{20} \times 20$$
$$\text{Mh} = 5$$

Nilai *orientation bin* sekarang:

0	0	15	15	0	0	0	0	0
0	20	40	60	80	100	120	140	160

3. Piksel [0, 2] mempunyai variabel r yang bernilai 165 sehingga nilai bl adalah 160, karena orientasi 160 kurang dari sama dengan r dan merupakan *orientasi bin* terdekat, sedangkan nilai bh adalah 180, karena tidak ada *orientation bin* yang lebih besar dari nilai r, maka nilai bh yang dipakai adalah nilai orientasi tertinggi ditambah nilai d yaitu 160 + 20 = 180. Proses *orientation binning* pada piksel [0, 2] adalah sebagai berikut:

Proses penghitungan besaran gradien Ml yang akan dimasukkan ke dalam *orientation bin* 160:

$$\text{Ml} = \frac{(\text{d} - (\text{r} - \text{bl}))}{\text{d}} \times \text{m}$$
$$\text{Ml} = \frac{(20 - (165 - 160))}{20} \times 30$$
$$\text{Ml} = \frac{15}{20} \times 30$$
$$\text{Ml} = 22{,}5$$

Nilai *orientation bin* sekarang:

0	0	15	15	0	0	0	0	22,5
0	20	40	60	80	100	120	140	160

Proses penghitungan besaran gradien Mh yang akan dimasukkan ke dalam *orientation bin* 60:

$$\text{Mh} = \frac{(\text{d} - (\text{bh} - \text{r}))}{\text{d}} \times \text{m}$$

$$\text{Mh} = \frac{(\text{d} - ((160 + \text{d}) - \text{r}))}{\text{d}} \times \text{m}$$

$$\text{Mh} = \frac{(20 - ((160 + 20) - 165))}{20} \times 30$$

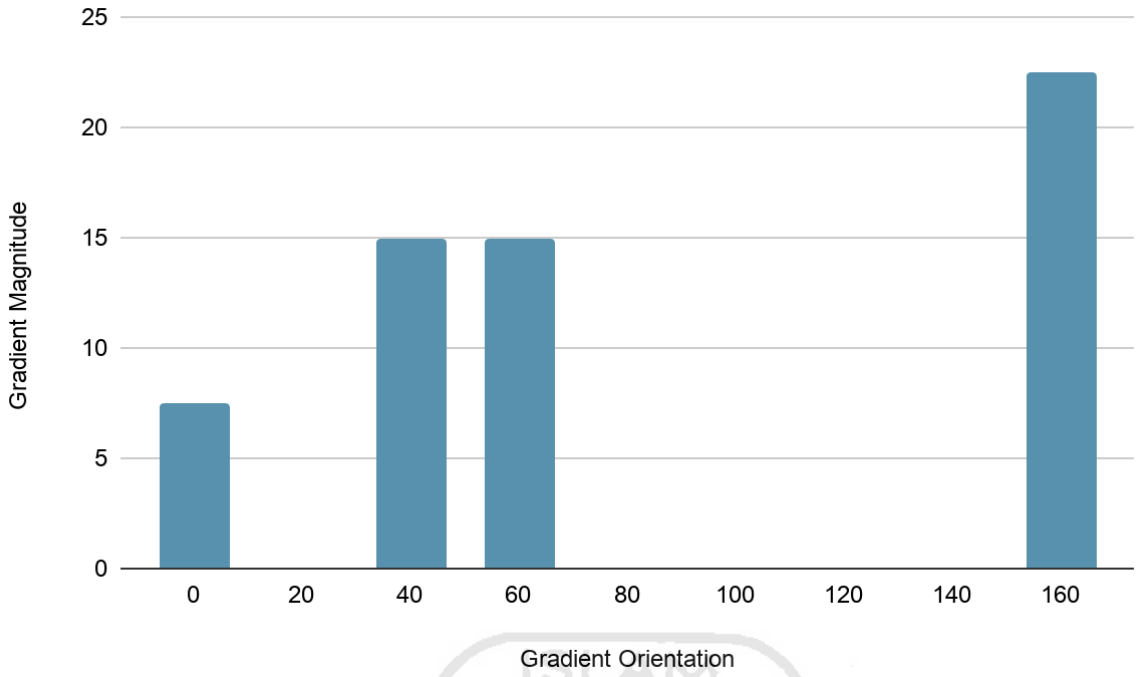
$$\text{Mh} = \frac{5}{20} \times 30$$

$$\text{Mh} = 7{,}5$$

Nilai *orientation bin* sekarang:

7,5	0	15	15	0	0	0	0	22,5
0	20	40	60	80	100	120	140	160

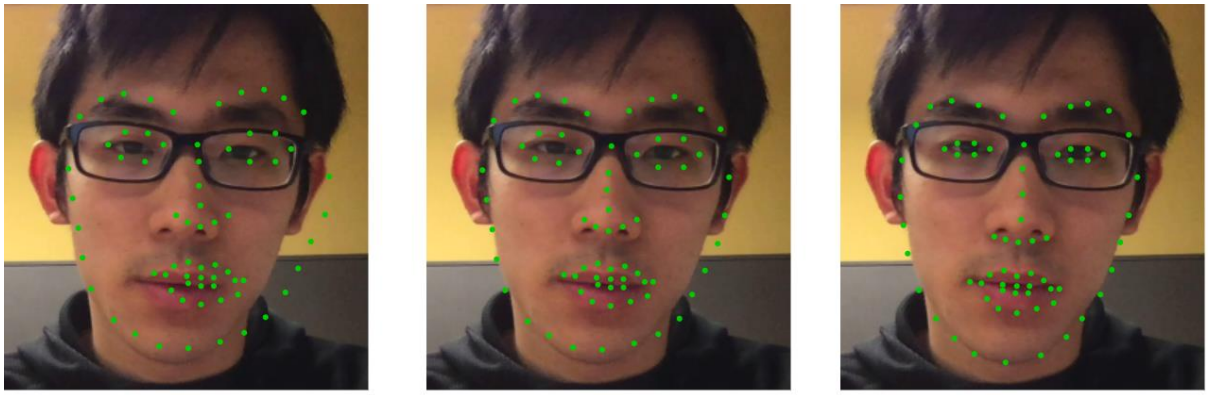
Hasil *orientation bin* ini dapat direpresentasikan menjadi histogram seperti pada Gambar 4.11. Oleh karena itu metode ini disebut sebagai *Histogram of Oriented Gradients*. *Histogram of Oriented Gradients* ini digunakan sebagai fitur dalam pengklasifikasian wajah dengan SVM.



Gambar 4.11 Histogram besaran gradien untuk tiap orientasi gradien

4.2.2 Pendeteksian Facial Landmark

Cara kerja pendeteksian *facial landmark* ini adalah dengan memanfaatkan rata-rata koordinat *facial landmark* dari suatu *dataset* dan *regressor* yang memprediksi *facial landmark* berdasarkan gambar wajah dan rata-rata koordinat *facial landmark*. Gambar 4.12 menunjukkan bagaimana *regressor* memprediksi *facial landmark* pada tiap iterasi dengan menggunakan rata-rata koordinat *facial landmark*. Variabel t pada Gambar 4.12 iterasi *regressor* melakukan prediksi. Pada nilai t = 0 atau iterasi ke-0, *regressor* menggunakan rata-rata koordinat *facial landmark* untuk memprediksi *facial landmark*. Pada iterasi ke-1, *regressor* menggunakan koordinat *facial landmark* hasil dari iterasi ke-0 untuk memprediksi koordinat *facial landmark* dan begitu seterusnya untuk iterasi berikutnya yaitu menggunakan koordinat *facial landmark* hasil iterasi sebelumnya untuk memprediksi hingga hasilnya dikatakan cukup akurat.



Gambar 4.12 Proses *regressor* memprediksi *facial landmark* pada tiap iterasi

4.2.3 Penghitugan EAR

Contoh penghitungan EAR pada Gambar 4.12 adalah sebagai berikut:

Misal:

p_1 = [20, 30], p_2 = [30, 20], p_3 = [60, 20], p_4 = [70, 30], p_5 = [60, 40], p_6 [30, 40]

Maka:

$$\text{EAR} = \frac{\|\overrightarrow{p_2} - \overrightarrow{p_6}\| + \|\overrightarrow{p_3} - \overrightarrow{p_5}\|}{2\|\overrightarrow{p_1} - \overrightarrow{p_4}\|}$$

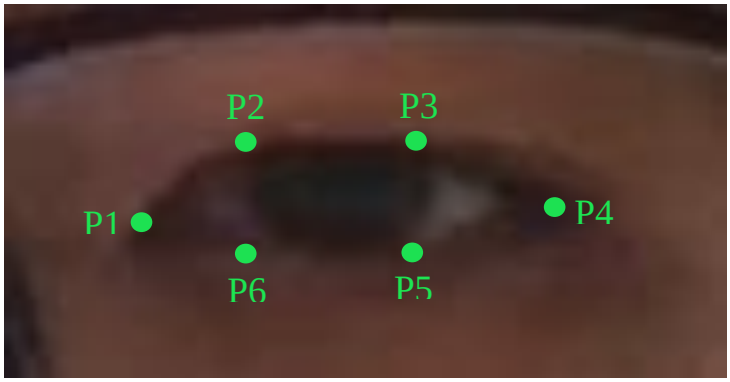
$$\text{EAR} = \frac{\sqrt{((x\overrightarrow{p_2} - x\overrightarrow{p_6})^2 + (y\overrightarrow{p_2} - y\overrightarrow{p_6})^2) + ((x\overrightarrow{p_3} - x\overrightarrow{p_5})^2 + (y\overrightarrow{p_3} - y\overrightarrow{p_5})^2)}}{2 \times \sqrt{(x\overrightarrow{p_1} - x\overrightarrow{p_4})^2 + (y\overrightarrow{p_1} - y\overrightarrow{p_4})^2}}$$

$$\text{EAR} = \frac{\sqrt{((30-30)^2 + (20-40)^2) + ((60-60)^2 + (20-40)^2)}}{2 \times \sqrt{(20-70)^2 + (30-30)^2}}$$

$$\text{EAR} = \frac{\sqrt{(0+400) + (0+400)}}{2 \times \sqrt{2500+0}}$$

$$\text{EAR} = \frac{28{,}28}{100}$$

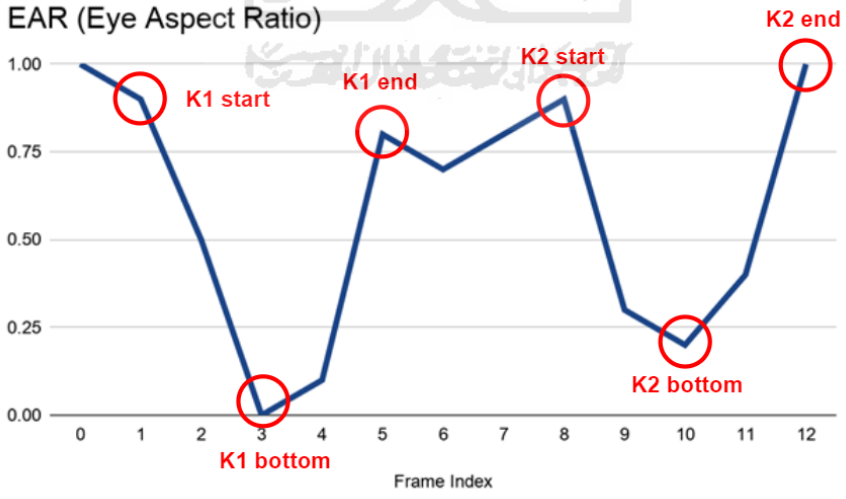
$$\text{EAR} = 0{,}2828$$



Gambar 4.13 Koordinat *facial landmark* pada mata

4.2.4 Pendeteksian Kedipan

Pendeteksian ini menggunakan SVM yang akan mengklasifikasi kedipan berdasarkan 13 EAR. Kedipan yang dideteksi oleh SVM akan diproses dengan algoritma pengambilan kedipan dari Ghoddoosian et al. (2019) untuk mendeteksi apakah terdapat lebih dari satu kedipan. Gambar 4.14 adalah grafik EAR untuk 13 *frame* yang mana pada rangkaian EAR tersebut terdeteksi 2 kedipan oleh algoritma pengambilan kedipan dari Ghoddoosian. Kedipan-kedipan yang terdeteksi pada Gambar 4.14 diberi nama K1 dan K2 dan kata *start*, *bottom*, dan *end* secara urut mempunyai arti yaitu mata terbuka pada awal siklus kedipan, mata tertutup, dan mata terbuka pada akhir siklus kedipan.



Gambar 4.14 Grafik nilai EAR yang terdiri dari 2 kedipan

Contoh cara kerja algoritma pengambilan kedipan dari Ghoddoosian untuk grafik pada Gambar 4.14 adalah sebagai berikut:

Misal:

$x[0] = 1$

$x[1] = 0{,}9$

$x[2] = 0{,}5$

$x[3] = 0$

$x[4] = 0{,}1$

$x[5] = 0{,}8$

$x[6] = 0{,}7;$

$x[7] = 0{,}8$

$x[8] = 0{,}9$

$x[9] = 0{,}3$

$x[10] = 0{,}2$

$x[11] = 0{,}4$

$x[12] = 1$

$\text{epsilon} = 0{,}01$

$M = 13$

Maka:

1: $\dot{x} \leftarrow x[n+1] - x[n],\ \forall\, n \in \{i | i = 0{,}1, \dots, M-2\}$

$\Rightarrow\ \dot{x} = [-0{,}1;\ -0{,}4;\ -0{,}5;\ 0{,}1;\ 0{,}7;\ -0{,}1;\ 0{,}1;\ 0{,}1;\ -0{,}6;\ -0{,}1;\ 0{,}2;\ 0{,}6]$

2: $\text{if } \dot{x}[0] = 0 \text{ then } \dot{x}[0] \leftarrow -1 \times \text{epsilon}$

$\Rightarrow\ \dot{x}[0] = -0{,}1$

3: $\dot{x}[n] \leftarrow \dot{x}[n-1] \times \text{epsilon},\ \forall\, n \in \{i |\ \dot{x}[i = 0 \wedge i \neq 0\}$

$\Rightarrow\ \dot{x} = [-0{,}1;\ -0{,}4;\ -0{,}5;\ 0{,}1;\ 0{,}7;\ -0{,}1;\ 0{,}1;\ 0{,}1;\ -0{,}6;\ -0{,}1;\ 0{,}2;\ 0{,}6]$

4: $c[n] \leftarrow \dot{x}[n+1] \times \dot{x}[n], \forall\, n \in \{\, i | i = 0{,}1, \dots, M-3\}$

$\Rightarrow\ c =$

$[0{,}04;\ 0{,}2;\ -0{,}05;\ 0{,}07;\ -0{,}07;\ -0{,}01;\ 0{,}01;\ -0{,}06;\ 0{,}06; -0{,}02;\ 0{,}12]$

5: $e \in \mathbb{R}^{P+2},\ P \leq M - 2$

6: $e[0] \leftarrow 0,\ e[P\ +\ 1] \leftarrow M - 1$

7: $e[k] \leftarrow n + 1,\ \forall\ (n \in \{i|c[i] < 0\} \wedge\ k \in \{i|I = 1, 2, \dots, P\})$

$\Rightarrow$ $e = [0;\ 3;\ 5;\ 6;\ 8;\ 10;\ 12]$

8: $THR\ \leftarrow\ 0.6\ \times \max(x) +\ 0.4\ \times \min(x)$

$\Rightarrow$ $THR = 0{,}6$

9: $t \in \mathbb{R}^{P+2}$

10: $t[0] \leftarrow +1,\ t[P + 1] \leftarrow +1$

11: Tambah +1 ke dalam t untuk setiap $n \in \{i|x[e[i]] > THR\}$ dan tambah -1 ke dalam t untuk setiap $n \in \{i|x[e[i]] \leq THR\}$, berurutan sesuai dengan indeks pada e

$\Rightarrow$ $t = [1;\ -1;\ \ 1;\ \ 1;\ \ 1;\ -1;\ \ 1]$

12: $z \in \mathbb{R}^{P+1},\ z[n]\ \leftarrow\ t[n + 1] \times t[n]$

$\Rightarrow$ $z = [-1;\ -1;\ \ 1;\ \ 1;\ -1;\ -1]$

13: s, untuk menyimpan semua indeks yang berisi nilai negatif di dalam z yang mempresentasikan arah bawah dan atas suatu pergerakan mata dalam kedipan

$\Rightarrow$ $s = [0;\ \ 1;\ \ 4;\ \ 5]$

14: $N\ \leftarrow\ \frac{\text{length}(s)}{2}$, N adalah banyak kedipan baru

$\Rightarrow$ $N = \frac{4}{2}$

$\Rightarrow$ $N = 2$

15: for $i \leftarrow 0$ to $N - 1$ do

16: $StartIndex \leftarrow e[s[2 \times i]]$

17: $EndIndex \leftarrow e[s[2 \times i + 1] + 1]$

18: $BottomIndex \leftarrow e[s[2 \times i + 1]]$

19: Hasil dari proses nomor 15 sampai 18 adalah 2 kedipan yang diberi nama K1 dan K2. Masing-masing mempunyai informasi StartIndex, EndIndex, dan BottomIndex. StartIndex adalah nomor *frame* saat mata terbuka di awal siklus kedipan. EndIndex adalah nomor *frame* saat mata terbuka di akhir siklus kedipan. BottomIndex adalah nomor *frame* saat mata tertup dalam siklus kedipan

⇒ K1:

⇒ StarIndex = 0

⇒ EndIndex = 3

⇒ BottomIndex = 5

⇒ K2:

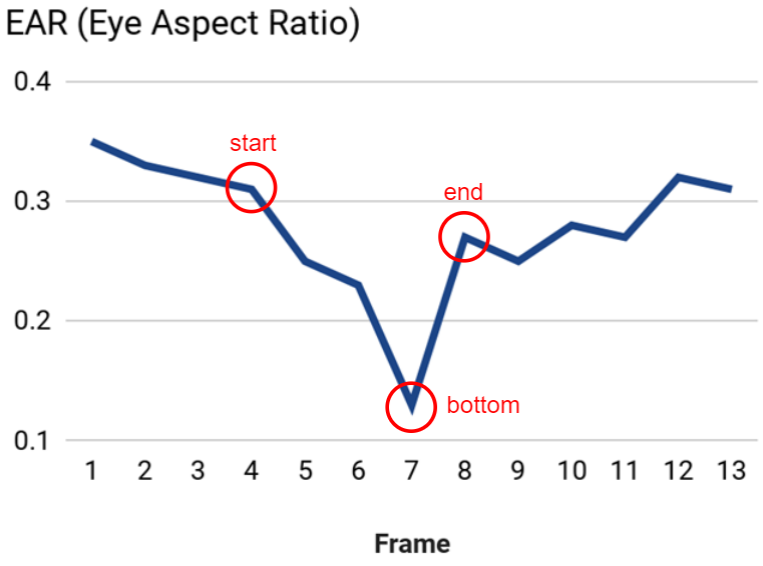
⇒ StarIndex = 8

⇒ EndIndex = 10

⇒ BottomIndex = 12

4.2.5 Penghitungan 4 Fitur Kedipan

Gambar 4.15 adalah grafik nilai EAR untuk 13 *frame*. Grafik tersebut merepresentasikan nilai-nilai EAR pada suatu kedipan. Berdasarkan grafik tersebut, kedipan dimulai dari *frame* ke 4 sampai ke *frame* 8.



Gambar 4.15 Grafik nilai EAR

Contoh penghitungan 4 fitur kedipan pada kedipan yang mempunyai 13 EAR seperti pada Gambar 4.15 adalah sebagai berikut:

Misal:

$start = 4, bottom = 7, end = 8$

$EAR[start_1] = 0{,}31; EAR[bottom_1] = 0{,}13;\ EAR[end_1] = 0{,}27$

$Number\ of\ blinks\ up\ to\ blink_1 = 2$

$Number\ of\ frames\ up\ to\ end_1 = 21$

Maka:

$Duration_1 = end_1 - start_1 + 1$

$Duration_1 = 8 - 4 + 1$

$Duration_1 = 5$

$$Amplitude_1 = \frac{EAR[start_1] - 2EAR[bottom_1] + EAR[end_1]}{2}$$

$$Amplitude_1 = \frac{0{,}31 - (2 \times 0{,}13) + 0{,}27}{2}$$

$Amplitude_1 = 0{,}16$

$$Eye\ Opening\ Velocity_1 = \frac{EAR[end_1]\ -\ EAR[bottom_1]}{end_1\ -\ bottom_1}$$

$$Eye\ Opening\ Velocity_1 = \frac{0{,}27\ -\ 0{,}13}{8\ -\ 7}$$

$Eye\ Opening\ Velocity_1 = 0{,}14$

$$Frequency_1 = 100\ \times \frac{Number\ of\ blinks\ up\ to\ blink_1}{Number\ of\ frames\ up\ to\ end_1}$$

$$Frequency_1 = 100\ \times \frac{2}{21}$$

$Frequency_1 = 9{,}52$

4.2.6 Preprocessing

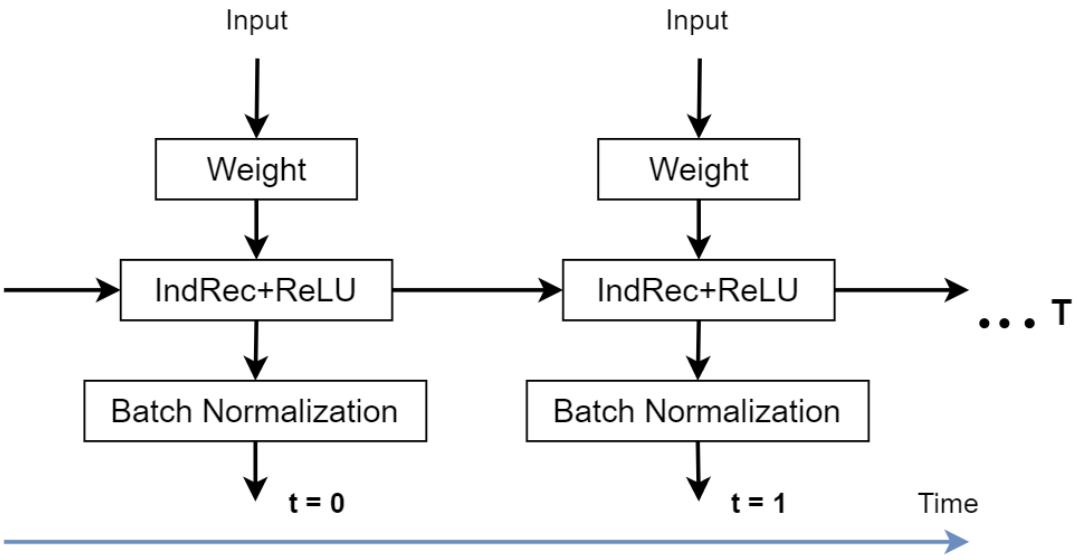
Setelah mendapatkan 4 fitur kedipan pada tahap sebelumnya, selanjutnya data tersebut diolah agar siap digunakan untuk *training* IndRNN. Pada penelitian ini, mengolah data dilakukan dengan cara normalisasi kemudian data tersebut dikelompokkan menjadi beberapa sekuens yang terurut sesuai nomor *frame*. Satu sekuens terdiri dari 30 data kedipan. Alasan nilai 30 kedipan ini dipilih sebagai jumlah kedipan dalam satu sekuens, agar jumlah data pada penelitian kali ini sama dengan jumlah data pada penelitian sebelumnya yaitu penelitian yang dilakukan oleh Ghoddoosian et al. (2019).

Kedipan seseorang dapat memiliki pola yang jauh berbeda antar individu sehingga *machine learning* sulit untuk menggeneralisasi, maka harus dilakukan normalisasi pada setiap kedipan seseorang (Ghoddoosian et al., 2019). Menurut Ghoddoosian et al. (2019), cara normalisasi fitur kedipan yaitu dengan menggunakan sepertiga fitur kedipan dalam keadaan tidak mengantuk untuk mendapatkan rata-rata dan standar deviasi setiap orang yang kemudian dimasukkan kedalam persamaan 4.2. Variabel $\overline{\text{Feature}}$ merupakan nilai fitur yang n adalah nomor fitur dan m nomor subjek. μ adalah rata-rata dan σ adalah standar deviasi.

$$\overline{\overline{\text{Feature}}}_{n,m} = \frac{\overline{\text{Feature}}_{n,m} - \mu_{n,m}}{\sigma_{n,m}} \qquad (4.2)$$

4.2.7 Pengklasifikasian dengan IndRNN

Arsitektur IndRNN ini terdiri dari 3 alur operasi, yang pertama yaitu pemberian bobot (*Weight*), mengaplikasikan fungsi IndRec+ReLu ke input, dan mengaplikasikan *batch normalization* (BN) untuk normalisasi input. Fungsi IndRec+ReLu adalah persamaan 3.12 yang sudah dijelaskan pada subbab 3.11. *Batch normalization* yaitu normalisasi bobot untuk mencegah sulitnya meningkatkan akurasi yang disebabkan distribusi data input yang bermasalah. Tiga alur operasi tersebut, dilakukan pada setiap variabel t, variabel t adalah data pada satu waktu atau satu elemen pada sekuens. Lalu terdapat variabel T yang menunjukkan seberapa banyak operasi IndRec+Relu dilakukan dalam satu *layer* dengan kata lain variabel T adalah jumlah dari variabel t. Gambar 4.16 adalah ilustrasi dari arsitektur IndRNN.



Gambar 4.16 Arsitektur IndRNN

Sebagaimana yang disebutkan pada subbab 4.2.6, satu sekuens terdiri dari 30 kedipan yang berarti T = 30, yang artinya IndRec+Relu dilakukan sebanyak 30 kali. Nilai t = 0 menunjukkan pemrosesan data kedipan pertama pada sekuens, t = 1 data kedipan kedua pada sekuens, dan begitu seterusnya sampai dengan t = 30.

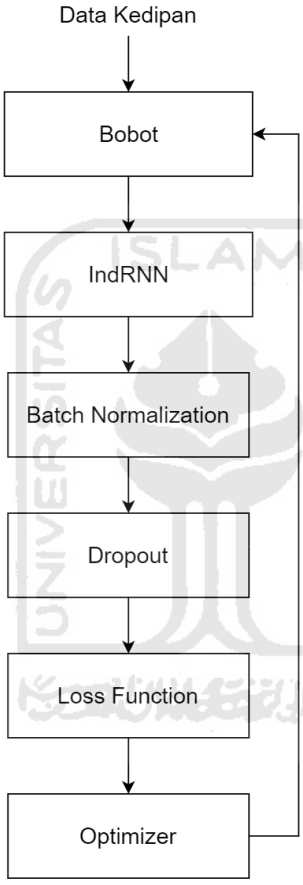
4.3 Training

Sebelum IndRNN dapat melakukan pengklasifikasian, IndRNN perlu di-*training* dengan *dataset* UTA-RLDD. Ilustrasi cara kerja *training* IndRNN dapat dilihat pada Gambar 4.17 yang penjelasannya adalah sebagai berikut:

i. Data kedipan diberi bobot yang merupakan representasi dari data kedipan.
j. Kemudian bobot tersebut dimasukkan kedalam IndRNN untuk diproses sehingga bobot dapat berubah menjadi representasi data yang lebih akurat dalam mengklasifikasi data temporal atau data kedipan.
k. Lalu operasi malakukan operasi *batch normalization* pada bobot.
l. Setelah operasi *batch normalization* dilakukan, *dropout* dilakukan untuk membuang bobot dari suatu neuron dan mencegah *overfitting* yang membuat model sulit menggeneralisasi.
m. Kemudian bobot dihitung skornya dengan *loss function*. *Loss function* adalah fungsi yang mengkalkulasi skor dari nilai bobot. Skor tersebut merupakan nilai untuk menunjukkan seberapa baik model tersebut mengklasifikasi. Semakin kecil skor maka

bobot tersebut akan semakin baik. Pada penelitian ini, *loss function* yang dipakai adalah *cross-entropy*. *Cross-entropy* adalah fungsi yang mengukur performa pengklasifikasian yang keluarannya berupa nilai diantara 0 sampai 1

n. Hasil skor tersebut digunakan sebagai bahan pertimbangan oleh o*ptimizer* untuk mengubah bobot sedemikian rupa sehingga model mempunyai akurasi yang lebih baik. *Optimizer* yang dipakai adalah *adam optimizer,* yang merupakan algoritma *stochastic gradient descent* yang dikembangkan khusus untuk *deep learning.*



Gambar 4.17 Cara kerja *training* IndRNN

Training IndRNN ini mempunyai 5 macam konfigurasi yaitu, *batch size, epoch, number of layers*, dan *dropout. Batch size* adalah seberapa banyak data yang akan diproses dalam satu proses *training*. *Epoch* adalah siklus *training* yang dimulai dari *training* data awal sampai data akhir dan setelah itu melakukan validasi data. *Number of layers* adalah seberapa banyak *hidden layer* yang digunakan oleh IndRNN. *Dropout* adalah nilai *dropout* yang digunakan untuk mencegah *overfitting*.

4.4 Pengujian

Pengujian ini dilakukan untuk mencari konfigurasi model IndRNN mana yang memiliki akurasi tertinggi pada data pengujian. Data pengujian ini adalah yang sudah melalui tahap *preprocessing*. Akurasi model IndRNN dihitung dengan cara menghitung berapa persen seluruh data pengujian dapat diklasifikasi dengan benar. Dengan contoh: semisal terdapat 10 data sekuens kedipan, dari 10 sekuens data tersebut, 7 sekuens data diklasifikasi dengan benar, sehingga nilai akurasinya adalah 70%.

4.5 Metode yang akan Dibandingkan

Implemenentai IndRNN ini akan dibandingkan akurasinya dengan implementasi HM-LSTM yang dibuat oleh Reza Ghoddoosian et al. (2019). HM-LSTM ini dimpelementasikan dengan *library* TensorFlow versi 1.5 dan bahasa pemrograman Python versi 3.6.9. Model HM-LSTM yang digunakan untuk perbandingan adalah model yang sudah di-*training* dengan data pengujian yang sama pada penelitian ini. Model HM-LSTM ini mempunyai 4 *layer*, tiap *layer* memiliki 32 *hidden state*.

BAB V
HASIL DAN PEMBAHASAN

5.1 Implementasi

Pendeteksian kantuk pada penelitian ini diimplementasikan dalam kode progam yang menggunakan bahasa pemrograman Python versi 3.7.8. Penjelasan kode program untuk tiap tahapan yang ada pada pendeteksian kantuk dijelaskan pada hasil dan pembahasan ini.

5.1.1 Pendeteksian Wajah

Sesuai dengan subbab 4.2.1 proses pendeteksian wajah dilakukan dengan 3 tahap yaitu normalisasi, hitung besaran dan orientasi gradien, dan *orientation binning*. Hasil implementasinya adalah sebagai berikut:

a. Normalisasi

Pada Gambar 5.1 menunjukkan kode *gamma correction* sesuai dengan persamaan 4.1 yang sudah penulis jelaskan sebelumnya pada subbab 4.2.1 poin a. Input parameter `frame` adalah gambar yang direpresentasikan dengan RGB, yang didalam kode program, RGB tersebut direpresentasikan dengan *array* 3 dimensi. *Array* 3 dimensi tersebut memuat 3 macam informasi, yaitu lebar gambar dalam piksel, tinggi gambar dalam pixel, dan nilai RGB untuk tiap piksel.

```
def adjust_gamma(frame, gamma=1.0):
    invGamma = 1.0 / gamma
    table = np.array([((i / 255.0) ** invGamma) * 255
                      for i in np.arange(0, 256)]).astype("uint8")

    return cv2.LUT(frame, table)
```

Gambar 5.1 Kode *gamma correction*

b. Proses Pendeteksian Wajah

Dalam pendeteksian wajah ini terdapat penghitungan besaran dan orientasi gradien serta *orientation binning* yang telah dijelaskan pada subbab 4.2.1 poin a dan b. Pendeteksi wajah ini adalah implementasi dari *library* dlib. *Library* dlib adalah *library* yang berisi kumpulan-kumpulan algoritma dan *tools machine learning*.

Video merupakan gabungan dari banyak *frame* yang berbeda-beda dan membentuk sebuah gambar yang bergerak. *Frame* inilah yang digunakan sebagai input pendeteksian wajah. Dalam kode program pada penelitian ini, *frame* ini dipresentasikan sama dengan *frame* yang telah disebutkan pada subbab 5.1.1 poina a. Sebelum *frame* dimasukkan ke dalam pendeteksian wajah, *frame* terlebih dahulu diubah lebarnya menjadi 550 piksel dan tinggi sesuai rasio ukuran *frame*, lalu *frame* tersebut diubah kecerahannya (normalisasi) dan warnanya diubah menjadi *grayscale*. Alasan kecerahan dan warna ini diubah karena implementasi pendeteksian wajah tersebut hanya bisa mendeteksi pada gambar *grayscale* dan pada kecerahan yang sudah dinormalisasi. Hasil pendeteksian wajah berupa koordinat dan ukuran kotak yang berisi wajah yang disebut *bounding box*. Kode tersebut dapat dilihat pada Gambar 5.2 dan hasil pendeteksian wajah dapat dilihat pada Gambar 5.3.

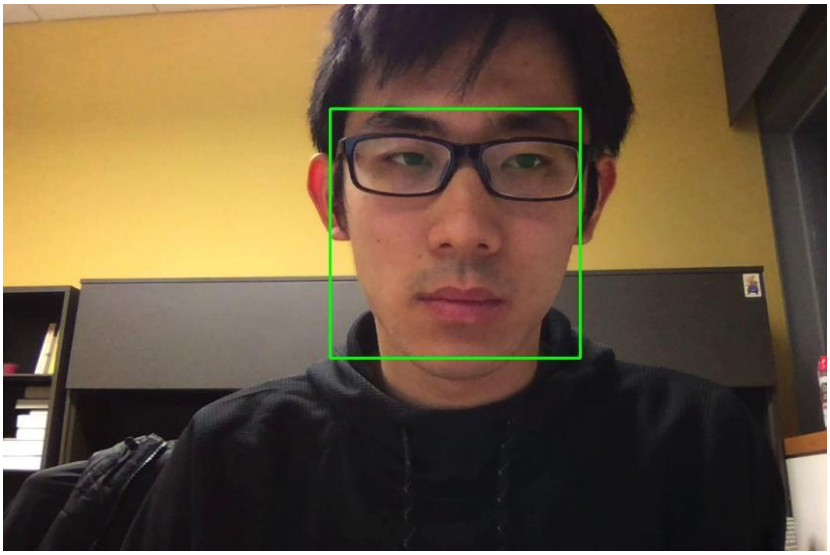
```
import dlib

frame = imutils.resize(frame, width=550)
rotation_degree = get_rotation(video_path)
frame = imutils.rotate_bound(frame, rotation_degree)

gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
gray = adjust_gamma(gray, gamma=1.5)

face_detector = dlib.get_frontal_face_detector()
rects = face_detector(gray, 0)
```

Gambar 5.2 Kode penghitungan besaran dan orientasi gradien serta *orientation binning*



Gambar 5.3 Hasil pendeteksian wajah

5.1.2 Pendeteksian Facial Landmark

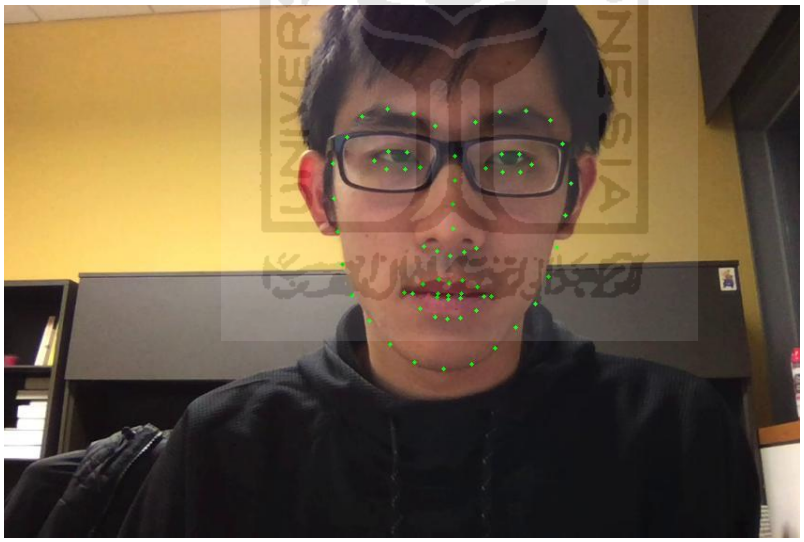
Sebagaimana halnya proses pendeteksian wajah, pendeteksian *facial landmark* juga menggunakan *library* dlib. Model yang digunakan oleh pendeteksi *facial landmark* ini adalah model yang sudah di-*training* dengan *dataset* dari Intelligent Behaviour Understanding Group (iBug) yang bernama iBUG 300-W *dataset*. Hasil pendeteskian *facial landmark* tersebut berupa 68 koordinat *facial landmark.* Untuk kode pendeteksian *facial landmark* dapat dilihat pada Gambar 5.4 dan hasil pendeteksian *facial landmark* dapat dilihat pada Gambar 5.5.

```
import dlib

facial_landmarks_predictor = dlib.shape_predictor(
    '/third_party_models/shape_predictor_68_face_landmarks.dat')

facial_landmarks = facial_landmarks_predictor(gray, rects[0])
facial_landmarks = face_utils.shape_to_np(facial_landmarks)
```

Gambar 5.4 Kode pendeteksian *facial landmark*



Gambar 5.5 Hasil pendeteksian *facial landmark*

5.1.3 Penghitungan EAR

Pada Gambar 5.6 menunjukkan kode penghitungan EAR untuk dua mata menggunakan persamaan 3.5 yang sudah penulis jelaskan sebelumnya pada subbab 3.5. Koordinat mata yang digunakan dalam penghitungan EAR ini, diambil dari hasil *facial landmark* sebelumnya, visualisasi koordinat mata ini dapat dilihat pada Gambar 5.7. Setelah EAR untuk dua mata didapatkan, hitung rata-rata dua EAR tersebut.

```
def get_left_eye_coordinates(facial_landmarks):
    (lStart, lEnd) = face_utils.FACIAL_LANDMARKS_IDXS["left_eye"]
    return facial_landmarks[lStart:lEnd]

def get_right_eye_coordinates(facial_landmarks):
    (rStart, rEnd) = face_utils.FACIAL_LANDMARKS_IDXS["right_eye"]
    return facial_landmarks[rStart:rEnd]

def eye_aspect_ratio(eye):
    A = dist.euclidean(eye[1], eye[5])
    B = dist.euclidean(eye[2], eye[4])

    C = dist.euclidean(eye[0], eye[3])

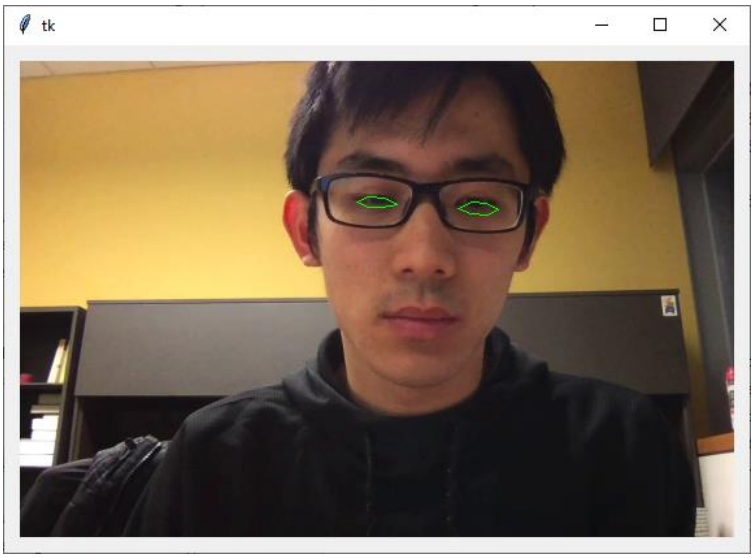
    ear = (A + B) / (2.0 * C)

    return ear

left_eye = get_left_eye_coordinates(facial_landmarks)
right_eye = get_right_eye_coordinates(facial_landmarks)
leftEAR = eye_aspect_ratio(left_eye)
rightEAR = eye_aspect_ratio(right_eye)
averaged_ear = (leftEAR + rightEAR) / 2.0
ear = averaged_ear

left_eye_hull = cv2.convexHull(left_eye)
right_eye_hull = cv2.convexHull(right_eye)
cv2.drawContours(frame, [left_eye_hull], -1, (0, 255, 0), 1)
cv2.drawContours(frame, [right_eye_hull], -1, (0, 255, 0), 1)
```

Gambar 5.6 Kode penghitungan EAR



Gambar 5.7 Wajah orang yang sedang berkedip

5.1.4 Pendeteksian Kedipan

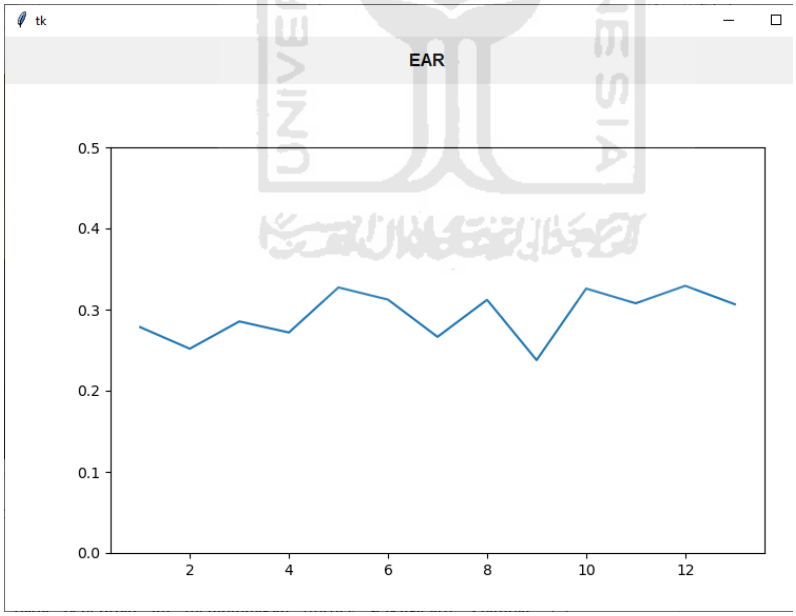
Sebagimana yang disebutkan pada subbab 4.2.4 implementasi pendeteksian kedipan ini menggunakan model linear SVM yang sudah disediakan oleh Ghoddoosian et al. (2019). Penerapan SVM pada penelitian ini menggunakan *library* scikit-learn. Gambar 5.9 menunjukkan penerapan SVM yang mendeteksi kedipan dalam rentang 13 *frame*. EAR dalam rentang 13 *frame* ini divisualisasikan seperti pada Gambar 5.10.

```
svm_blink_detector = pickle.load(open(
'D:/ProjectApps/work/drowsiness_detector/third_party_models/Trained_SVM_C
=1000_gamma=0.1_for 7kNegSample.sav', 'rb'))

EAR_series = shift(EAR_series, -1, cval=ear)
is_blink = svm_blink_detector.predict(EAR_series.reshape(1,-1))

gui.update_ear(EAR_series)
```

Gambar 5.8 Kode penerapan SVM



Gambar 5.9 Grafik EAR pada rentang 13 *frame*

Setelah dideteksi dengan linear SVM, kemudian data tersebut dicek dan diperbaiki menggunakan algoritma pengambilan kedipan Ghoddoosian et al. (2019). Tujuan algoritma tersebut adalah memastikan tidak ada serangkaian kedipan yang dianggap sebagai satu

kedipan. Gambar 5.10 menunjukkan implementasi algoritma Ghoddoosian et al. yang telah dijelaskan pada subbab 3.6.

```
retrieved_blinks = []
missed_blinks = False
values = np.asarray(detected_blink.values)
threshold = 0.4 * np.min(values) + 0.6 * np.max(values)
N = len(values)
derivative = values[1:N] - values[0:N-1]
i = np.where(derivative == 0)
if len(i[0]) != 0:
    for k in i[0]:
        if k == 0:
            derivative[0] = -EPSILON
        else:
            derivative[k] = EPSILON * derivative[k-1]
M = N - 1
zero_crossing = derivative[1:M] * derivative[0:M-1]
x = np.where(zero_crossing < 0)
xtrema_index = x[0] + 1
xtrema_EAR = values[xtrema_index]
updown = np.ones(len(xtrema_index))
updown[xtrema_EAR < threshold] = -1

updown = np.concatenate(([1],updown,[1]))
xtrema_EAR = np.concatenate(([values[0]],xtrema_EAR,[values[N - 1]]))
xtrema_index = np.concatenate(([0], xtrema_index,[N - 1]))
updown_xero_crossing = updown[1:len(updown)] * updown[0:len(updown) - 1]
jump_index = np.where(updown_xero_crossing < 0)
number_of_blinks = int(len(jump_index[0]) / 2)
selected_EAR_First = xtrema_EAR[jump_index[0]]
selected_EAR_Sec = xtrema_EAR[jump_index[0] + 1]
selected_index_First = xtrema_index[jump_index[0]]
selected_index_Sec = xtrema_index[jump_index[0] + 1]
if number_of_blinks > 1:
    missed_blinks = True
if number_of_blinks == 0:
    print(updown, detected_blink.duration)
    print(values)
    print(derivative)
for j in range(number_of_blinks):
    detected_blink = Blink()
    detected_blink.start = selected_index_First[2 * j]
    detected_blink.peak = selected_index_Sec[2 * j]
    detected_blink.end = selected_index_Sec[2 * j + 1]
    detected_blink.startEAR = selected_EAR_First[2 * j]
    detected_blink.peakEAR = selected_EAR_Sec[2 * j]
```

```
    detected_blink.endEAR = selected_EAR_Sec[2 * j + 1]
    detected_blink.duration = detected_blink.end - detected_blink.start +
1
    detected_blink.amplitude = 0.5 * (detected_blink.startEAR-
detected_blink.peakEAR) + 0.5 * (detected_blink.endEAR-
detected_blink.peakEAR)
    detected_blink.velocity = (detected_blink.endEAR -
selected_EAR_First[2 * j + 1]) / (detected_blink.end -
selected_index_First[2 * j + 1] + 1)
    retrieved_blinks.append(detected_blink)
return missed_blinks,retrieved_blinks
```

Gambar 5.10 Kode implementasi algoritma Ghoddoosian et al.

5.1.5 Penghitungan 4 Fitur Kedipan

Setelah itu, kedipan yang terdeteksi dihitung keempat nilai fiturnya yaitu durasi, amplitudo, kecepatan mata membuka dan frekuensinya. Implementasi algoritma yang dijelaskan pada subbab 3.7 dapat dilihat pada Gambar 5.11.

```
class BlinkDetector:
    def ultimate_blink_check(self, detected_blink):
        detected_blink.duration = detected_blink.end -
detected_blink.start + 1
        detected_blink.amplitude = 0.5 * (detected_blink.startEAR-
detected_blink.peakEAR) + 0.5 * (detected_blink.endEAR-
detected_blink.peakEAR)
        detected_blink.velocity = (detected_blink.endEAR -
selected_EAR_First[2 * j + 1]) / (detected_blink.end -
selected_index_First[2 * j + 1] + 1)

def blink_detector(output_textfile, video_path):

    retrieved_blinks = blink_detector.track_ears(bd_request)

    if retrieved_blinks:
        total_blinks = blink_detector.get_total_blinks()
        blink_frame_freq = total_blinks / number_of_frames
```

Gambar 5.11 Kode penghitungan 4 fitur kedipan

5.1.6 Preprocessing

Sebagaimana dijelaskan pada subbab 4.2.6, *preprocessing* dilakukan dengan 2 tahap yaitu normalisasi dan pengelompokkan data kedipan. Untuk implementasi normalisasi dapat dilihat pada Gambar 5.12. Gambar 5.12 menunjukkan normalisasi untuk setiap fitur kedipan.

```
def normalize_blinks(self, blinks_count,
    freq, freq_mean, freq_stddev,
    amp, amp_mean, amp_stddev,
    dur, dur_mean, dur_stddev,
    vel, vel_mean, vel_stddev
):
    normalized_blinks = np.zeros([blinks_count, 4])
    normalized_freq = (freq[0:blinks_count] - freq_mean) / freq_stddev
    normalized_blinks[:, FREQ_INDEX] = normalized_freq
    normalized_amp = (amp[0:blinks_count]  - amp_mean) / amp_stddev
    normalized_blinks[:, AMP_INDEX] = normalized_amp
    normalized_dur = (dur[0:blinks_count]  - dur_mean) / dur_stddev
    normalized_blinks[:, DUR_INDEX] = normalized_dur
    normalized_vel = (vel[0:blinks_count]  - vel_mean)  / vel_stddev
    normalized_blinks[:, VEL_INDEX] = normalized_vel
    return normalized_blinks
```

Gambar 5.12 Kode implementasi normalisasi data kedipan

Gambar 5.13 adalah implementasi dari pengelompokkan data kedipan menjadi beberapa sekuens. Pengelompokkan menjadi beberapa sekuens ini berdasarkan tiap langkah. Contoh pengelompokkan menjadi beberapa sekuens berdasarkan tiap langkah yaitu misal terdapat 34 data kedipan, satu sekuens terdiri dari 30 data, dan dikelompokkan tiap 2 langkah, maka hasil pengelompokkan tersebut adalah 3 sekuens kedipan yaitu data kedipan ke-1 sampai 30, data kedipan ke-3 sampai 32, dan data kedipan ke-5 sampai 34.

```
def unroll_in_time(in_data, window_size, stride):
    n = len(in_data)
    if n <= window_size:
        out_data = np.zeros([1, window_size, 4])
        out_data[0, -n:, :] = in_data
        return out_data
    else:
        N = ((n - window_size) // stride) + 1
        out_data = np.zeros([N, window_size, 4])
        for i in range(N):
            if i * stride + window_size <= n:
                out_data[i, :, :] = in_data[i * stride:i * stride +
window_size, :]
            else:
                break

        return out_data
```

Gambar 5.13 Kode implementasi pengelompokkan data kedipan

5.1.7 Pengklasifikasian dengan IndRNN

Implementasi IndRNN ini menggunakan *library* PyTorch versi 1.6.0 dalam versi Python 3.7.8. Gambar 5.14 menunjukkan implementasi IndRNN berdasarkan arsitektur yang sudah dijelaskan pada subbab 4.2.7.

```
class IndrnnPlainnet(nn.Module):
    def __init__(self, input_size, outputclass):
        super(IndrnnPlainnet, self).__init__()
        hidden_size = args.hidden_size

        self.dense_input_list = nn.ModuleList()
        dense_input = Linear_overtime(input_size, hidden_size)
        self.dense_input_list.append(dense_input)
        for x in range(args.num_layers - 1):
            dense_input = Linear_overtime(hidden_size, hidden_size)
            self.dense_input_list.append(dense_input)

        self.RNNs = nn.ModuleList()
        for x in range(args.num_layers):
            rnn = IndRNNwithBN(hidden_size=hidden_size,
seq_len=args.seq_len)
            self.RNNs.append(rnn)

        self.classifier = nn.Linear(hidden_size, outputclass, bias=True)
        self.init_weights()

    def forward(self, input):
        rnnoutputs = {}
        rnnoutputs['outlayer-1'] = input
        for x in range(len(self.RNNs)):
            rnnoutputs['dilayer%d' % x] =
self.dense_input_list[x](rnnoutputs['outlayer%d' % (x - 1)])
            rnnoutputs['outlayer%d' % x] =
self.RNNs[x](rnnoutputs['dilayer%d' % x])
            if args.dropout > 0:
                rnnoutputs['outlayer%d' % x] = dropout_overtime(

rnnoutputs['outlayer%d'%x],args.dropout,self.training)
        temp = rnnoutputs['outlayer%d' % (len(self.RNNs) - 1)][-1]
        output = self.classifier(temp)
        return output
```

Gambar 5.14 Kode implementasi IndRNN

5.2 Training

Implementasi *training* yang sudah dijelaskan pada subbbab 4.3 dapat dilihat pada Gambar 5.15. Hasil dari penjalanan kode *training* ini dapat dilihat pada Gambar 5.16. Gambar 5.16 menunjukkan bagaimana proses *training* untuk tiap *epoch*, yang setiap *epoch* menampilkan akurasi hasil dari evaluasi dengan data evaluasi. Model akurasi tertinggi dari seluruh *epoch* inilah yang akan dihasilkan oleh kode *training* Gambar 5.15. *Training* ini dijalankan pada Google Colab dengan GPU Nvidia K80s.

```
for epochi in range(num_epoch):
    train(num_train_batches)
    acc = eval(dh_eval, num_eval_batches)

    print('>>>>>>>>>>>>>>>> Epoch {0} <<<<<<<<<<<<<<<<'.format(epochi +
1))
    print('Accuracy: ', acc)
    print()

    if (acc > last_high_acc):
        best_model = copy.deepcopy(model.state_dict())
        best_opti = copy.deepcopy(optimizer.state_dict())

        prev_acc = last_high_acc
        last_high_acc = acc
        epoch_with_high_acc = epochi + 1

        print('====== CHANGE CURRENT HIGHEST ACCURACY ======')
        print('Previous Highest Accuracy  : {0}'.format(prev_acc))
        print('Current Highest Accuracy   : {0}'.format(last_high_acc))
        print('\n')

        patience = 0
    elif (patience > patience_threshold):
        model.load_state_dict(best_model)
        optimizer.load_state_dict(best_opti)
    else:
        patience += 1

model.load_state_dict(best_model)
optimizer.load_state_dict(best_opti)
```

```
elapsed = time.time() - start_time

test_acc = test(dh_test)

print('')
print('====== FINAL HIGHEST ACCURACY ======')
print('Epoch                          : {0}'.format(epoch_with_high_acc))
print('Test Accuracy                  : {0}'.format(test_acc))
print('Training Time                  : {0} seconds'.format(elapsed))
print('')
```

Gambar 5.15 Kode implementasi *training* IndRNN

```
>>>>>>>>>>>>>>>> Epoch 77 <<<<<<<<<<<<<<<<
Accuracy:  0.89375

>>>>>>>>>>>>>>>> Epoch 78 <<<<<<<<<<<<<<<<
Accuracy:  0.890625

>>>>>>>>>>>>>>>> Epoch 79 <<<<<<<<<<<<<<<<
Accuracy:  0.896875

>>>>>>>>>>>>>>>> Epoch 80 <<<<<<<<<<<<<<<<
Accuracy:  0.884375


====== FINAL HIGHEST ACCURACY ======
Epoch                          : 29
Test Accuracy                  : 0.8921713441654358
Training Time                  : 32.6152925491333 seconds
```

Gambar 5.16 Hasil dari penjalanan kode *training*

5.3 Pengujian

Pengujian dilakukan pada Google Colab dengan GPU Nvidia K80s. Hasil pengujian model dengan berbagai macam variasi konfigurasi dapat dilihat pada Tabel 5.1 dan berdasarkan hasil tersebut didapatkan konfigurasi model yang memiliki akurasi terbesar, konfigurasinya yaitu *batch size* 64, *epoch* 80, *number of layers* 2, *hidden size* 512, dan *dropout* 0.5 dengan akurasi 91,88% dan lama *training* 53 detik.

Tabel 5.1 Hasil pengujian model

No.	Batch Size	Epoch	Number of Layers	Hidden Size	Dropout	Akurasi	Lama Training (detik)
1	20	80	4	512	0,5	88,96%	214
2	32	80	4	512	0,5	90,21%	139
3	48	80	4	512	0,5	91,25%	122
4	64	80	4	512	0,5	90,93%	111
5	80	80	4	512	0,5	88,95%	105
6	128	80	4	512	0,5	90,14%	98
7	160	80	4	512	0,5	89,48%	92
8	192	80	4	512	0,5	90%	92
9	64	20	4	512	0,5	88,96%	28
10	64	32	4	512	0,5	89,47%	44
11	64	48	4	512	0,5	90,1%	67
12	64	72	4	512	0,5	90,52%	101
13	64	96	4	512	0,5	90,42%	134
14	64	120	4	512	0,5	90,31%	168
15	64	160	4	512	0,5	90,83%	225
16	64	200	4	512	0,5	91,46%	280
17	64	80	1	512	0,5	90,83%	26
18	64	80	2	512	0,5	91,88%	53
19	64	80	6	512	0,5	87,5%	170
20	64	80	8	512	0,5	87,71%	228
21	64	80	12	512	0,5	82,5%	347
22	64	80	16	512	0,5	69,38%	463
23	64	80	4	8	0,5	81,15%	44
24	64	80	4	32	0,5	88,96%	43
25	64	80	4	64	0,5	88,54%	67
26	64	80	4	128	0,5	89,69%	70
27	64	80	4	256	0,5	90,42%	70
28	64	80	4	1024	0,5	91,56%	245

29	64	80	4	1536	0,5	90,83%	431
30	64	80	4	2048	0,5	90,73%	667
31	64	80	4	512	0,1	89,79%	111
32	64	80	4	512	0,3	90,21%	111
33	64	80	4	512	0,7	87,4%	111
34	64	80	4	512	0,9	82,92%	111
35	64	80	4	512	1	31,25%	111

Dari Tabel 5.1 dapat disimpulkan bahwa *batch size* hampir tidak mempengaruhi akurasi tapi mempengaruhi lamanya waktu training, semakin kecil *batch size*, maka semakin singkat waktunya. *Epoch* hampir tidak mempengaruhi akurasi, setidaknya akurasi 20 *epo*ch tidak jauh beda dengan *epoch* lain yang lebih besar dan *epoch* mempengaruhi lamanya waktu *training* dengan cukup siginifikan, semakin banyak *epoch*, maka semakin lama waktunya. *Number of layers* dapat mengurangi akurasi secara signifikan jika *number of layers* lebih dari sama dengan 12. Semakin banyak *number of layers*, maka semakin lama waktu *training*-nya. *Hidden size* dapat mengurangi akurasi jika *hidden size* bernilai di bawah 32 dan dapat mempengaruhi lamanya waktu training, semakin besar *hidden size*, maka semakin lama waktunya. *Dropout* dapat mengurangi akurasi secara signifikan jika *dropout* bernilai lebih dari 0,5.

Dari 35 pengujian, sebanyak 18 pengujian memperoleh tingkat akurasi di atas 90% dan 33 pengujian memperoleh tingkat akurasi di atas 80%. Hal ini menunjukkan, IndRNN tidak perlu pengujian konfigurasi yang banyak untuk mendapatkan akurasi di atas 80%.

5.4 Perbandingan Hasil Implementasi

Hasil pengujian implementasi HM-LSTM dengan data pengujian dan dilakukan pada Google Colab dengan GPU Nvidia K80s, menunjukkan HM-LSTM mempunyai akurasi 86,48% dan lama *training* 670 detik. Jadi bisa disimpulkan IndRNN lebih akurat dan lebih cepat dari HM-LSTM.

BAB VI
PENUTUP

6.1 Kesimpulan

Berdasarkan hasil penelitian tentang implementasi *deep learning* untuk pendeteksian kantuk berdasarkan fitur kedipan dapat disimpulkan bahwa:

1. IndRNN memiliki akurasi yang lebih baik dari pada HM-LSTM. IndRNN memiliki akurasi 91,88% dan HM-LSTM memiliki akurasi 86,48%.
2. Waktu yang dibutuhkan IndRNN untuk *training* dengan 80% *dataset* UTA-RLDD yaitu 53 detik, lebih cepat dibandingkan dengan HM-LSTM yang membutuhkan waktu 670 detik untuk *training* dengan *dataset* yang sama.

6.2 Saran

Saran yang dapat ditulis untuk penelitian ini adalah:

1. Diharapkan pada penelitian selanjutnya, *dataset* yang digunakan untuk *training* dan pengujian lebih banyak dari penelitian ini, untuk memastikan apakah IndRNN memiliki performa yang lebih bagus atau kurang bagus pada *dataset* yang sangat banyak.

DAFTAR PUSTAKA

Arfan, I., & Wulandari. (2018). Studi Epidemiologi Kejadian Kecelakaan Lalu Lintas di Pontianak. *Jurnal Vokasi Kesehatan, 4(2).*

Belakhdar, I., Kaaniche, W., Djmel, R., & Ouni, B. (2016, 21 Maret). *Detecting driver drowsiness based on single electroencephalography channel*. 2016 13th International Multi-Conference on Systems, Signals & Devices (SSD), Leipzig, Germany. https://doi.org/10.1109/SSD.2016.7473671

Chandrasena, H. M., & Wickramasinghe, D. M. J. (2016). Driver's Drowsiness Detecting and Alarming System. *International Journal of Computer Science and Information Technology Research, 4(3),* 127-139.

Chirra, V. R. R., Uyyla, S. R., & Kolli, V. K. K. (2019). Deep CNN: A Machine Learning Approach for Driver Drowsiness Detection Based on Eye State. *Revue d'Intelligence Artificielle,* 33(6), 461-466.

Chollet, F. (2017). *Deep Learning with Python.* Shelter Island: Manning.

Chung, J., Ahn, S., & Bengio, Y. (2017). *Hierarchical Multiscale Recurrent Neural Networks.* International Conference on Learning Representations 2017, Toulon, France.

Dalal, N., & Triggs, B. (2005, 20 Juni). *Histograms of oriented gradients for human detection*. 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05), San Diego, CA, USA. https://doi.org/10.1109/CVPR.2005.177

Friedrichs, F., & Yang, B. (2010, 21 Juni). *Camera-based Drowsiness Reference for Driver State Classification under Real Driving Conditions.* 2010 IEEE Intelligent Vehicles Symposium, San Diego, CA, USA. https://doi.org/10.1109/IVS.2010.5548039

Ghoddoosian, R., Galib, M., & Athitsos, V. (2019, 16 Juni). *A Realistic Dataset and Baseline Temporal Model for Early Drowsiness Detection*. 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW), Long Beach, CA, USA, USA. https://doi.org/10.1109/CVPRW.2019.00027

Jiao, Y., Deng, Y., Luo, Y., & Lu, B. (2020). Driver sleepiness detection from EEG and EOG signals using GAN and LSTM networks. *Neurocomputing,* 408, 100-111.

Kazemi, V., & Sullivan, J. (2014, 23 Juni). *One millisecond face alignment with an ensemble of regression trees*. 2014 IEEE Conference on Computer Vision and Pattern Recognition, Columbus, OH, USA. https://doi.org/10.1109/CVPR.2014.241

Li, S., Li, W., Cook, C., Zhu, C., & Gao, Y. (2018, 18 Juni). *Independently Recurrent Neural Network (IndRNN): Building A Longer and Deeper RNN*. 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, Salt Lake City, UT, USA. https://doi.org/10.1109/CVPR.2018.00572

Nugroho, A. S., Witarto, A. B., & Handoko, D. (2003). Support Vector Machine: Teori dan Aplikasinya dalam Bioinformatika. IlmuKomputer.com

Pienaar, S. W., & Malekian, R. (2019, 18 Agustus). *Human Activity Recognition Using LSTM-RNN Deep Neural Network Architecture.* 2019 IEEE 2nd Wireless Africa Conference (WAC), Pretoria, South Africa, South Africa. https://doi.org/10.1109/AFRICA.2019.8843403

Soukupová, T., & Čech, J. (2016, 3 Februari). *Real-Time Eye Blink Detection using Facial Landmarks*. 21st Computer Vision Winter Workshop, Rimske Toplice, Slovenia.

Tadesse, E., Sheng, W., & Liu, M. (2014, 31 Mei). *Driver drowsiness detection through HMM based dynamic modeling*. 2014 IEEE International Conference on Robotics and Automation (ICRA), Hong Kong, China. https://doi.org/10.1109/ICRA.2014.6907440

Vural, E., Çetin, M., Erçil, A., Littlewort, G., Bartlett, M., & Movellan, J. (2009). Machine Learning Systems for Detecting Driver Drowsiness. Boston, MA: Springer.

Willy, T. DR. (2019, Juli 12). Kantuk. Alodokter. https://www.alodokter.com/kantuk