SKRIPSI

PENDETEKSIAN API BERBASIS PENGOLAHAN CITRA DIGITAL MENGGUNAKAN METODE *CONVOLUTIONAL NEURAL NETWORK*

DIGITAL IMAGE PROCESSING - BASED FIRE DETECTION USING CONVOLUTIONAL NEURAL NETWORK METHOD

MOHAMAD INDRA WICAKSONO
13/348561/PA/15460

PROGRAM STUDI ILMU KOMPUTER
DEPARTEMEN ILMU KOMPUTER DAN ELEKTRONIKA
FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM
UNIVERSITAS GADJAH MADA
YOGYAKARTA

2020

SKRIPSI

PENDETEKSIAN API BERBASIS PENGOLAHAN CITRA DIGITAL MENGGUNAKAN METODE *CONVOLUTIONAL NEURAL NETWORK*

DIGITAL IMAGE PROCESSING - BASED FIRE DETECTION USING CONVOLUTIONAL NEURAL NETWORK METHOD

Diajukan untuk memenuhi salah satu syarat memperoleh derajat
Sarjana Ilmu Komputer

MOHAMAD INDRA WICAKSONO
13/348561/PA/15460

PROGRAM STUDI ILMU KOMPUTER
DEPARTEMEN ILMU KOMPUTER DAN ELEKTRONIKA
FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM
UNIVERSITAS GADJAH MADA
YOGYAKARTA

2020

HALAMAN PENGESAHAN

SKRIPSI

PENDETEKSIAN API BERBASIS PENGOLAHAN CITRA DIGITAL MENGGUNAKAN METODE *CONVOLUTIONAL NEURAL NETWORK*

Telah dipersiapkan dan disusun oleh

MOHAMAD INDRA WICAKSONO

13/348561/PA/15460

Telah dipertahankan di depan Tim Penguji

pada tanggal 20 Januari 2020

Pembimbing
Wahyono, Ph.D

Penguji 1
Suprapto, Drs. M.Kom., Dr.

Mengetahui
a.n. Dekan FMIPA-UGM
Wakil Dekan Bidang Akademik dan Kerjasama Mahasiswaan

Dr.rer.nat. Nurul Hidayat Aprilia, M.Si
NIP. 197304071998031002

Penguji 2
Moh Edi Wibowo, S.Kom., M.Kom., Ph.D

PERNYATAAN BEBAS PLAGIASI

Saya yang bertanda tangan di bawah ini :

Nama	: Mohamad Indra Wicaksono
NIM	: 13/348561/PA/15460
Tahun Terdaftar	: 2013
Program Studi	: Ilmu Komputer
Fakultas	: Matematika dan Ilmu Pengetahuan Alam

Menyatakan bahwa dalam dokumen ilmiah Skripsi ini tidak terdapat bagian dari karya ilmiah lain yang telah diajukan untuk memperoleh gelar akademik di suatu Lembaga Pendidikan Tinggi, dan juga tidak terdapat karya atau pendapat yang pernah ditulis atau diterbitkan oleh orang/ Lembaga lain, kecuali secara tertulis disitasi dalam dokumen ini dan disebutkan sumbernya secara lengkap dalam daftar pustaka.

Dengan demikian saya menyatakan bahwa dokumen ilmiah ini bebas dari unsur-unsur plagiasi dan apabila dokumen ilmiah Skripsi ini di kemudian hari terbtuki merupakan hasil plagiasi dari hasil karya penulis lain dan/atau dengan sengaja mengajukan karya atau pendapat yang merupakan hasil karya penulis lain, maka penulis bersedia menerima sanksi akademik dan/atau sanksi hukum berlaku.

Yogyakarta, 18-02-2020

Mohamad Indra Wicaksono
13/348561/PA/15460

PRAKATA

Puji syukur kehadirat Allah SWT atas limpahan rahmat, karunia, serta petunjuk-Nya sehingga tugas akhir berupa penyusunan skripsi ini telah terselesaikan dengan baik.

Dalam penyusunan tugas akhir ini penulis telah banyak mendapatkan arahan, bantuan, doa serta dukungan dari berbagai pihak. Oleh karena itu pada kesempatan ini penulis mengucapan terima kasih kepada:

1. Bapak Wahyono, Ph.D selaku dosen pembimbing skripsi yang berkenan meluangkan waktu dan pikiran dalam penyelesaian tugas akhir.
2. Bapak, ibu dan kedua adik saya yang selalu mendukung saya untuk menyelesaikan kuliah.
3. Segenap Dosen dan civitas akademik di lingkungan program Studi Ilmu Komputer, Jurusan Ilmu Komputer dan Elektronika, Fakultas Matematika dan Ilmu Pengetahuan Alam, Universitas Gadjah Mada.
4. Bu Aina Musdholifah, S.Kom., M.Kom., Ph.D selaku dosen pembimbing akademik.
5. Achmad Jeihan P., Muhammad Ridwan A.B., Prawira Jalu N. dan Rozan Ro'if., teman-teman program studi Ilmu Komputer 2013.
6. Laila Annafi, Safira J. Azarina, Taradhinta S., Swasti Retno T.H., dan Dynni Puspitasari, teman-teman KKN JTG-55.
7. Denny Yusuf W., Fawwaz Friyan P., Dandy N. Fauzan, dan Rilo Dwi K. teman-teman SMP Negeri 1 Tangerang.
8. Winston Wilford, Archie Ardhana, dan M. Dennis Kananda, teman-teman SMA Harapan Bangsa.
9. Kakak Beriya Phoa, guru dan mentor sejak SMA.
10. M. Antoni, Fariz Marshall W., M. Arief R, Narahadi Muhammad, dan Farah Arinda Kirana, Teman-teman A121 Universiti Utara Malaysia.
11. Mas Novi, Mas Rio, Mas Alex, Mas Aris, Mas Antoni, Mas Alter, Mas Djarot, Mas Franky, Mas Caesar, dan Mas Firdaus, keluarga IBM Indonesia.
12. Seluruh keluarga besar Eyang Surono dan Mbah Soetarman.
13. Staff Mataram Fotokopi yang mencetak semua naskah proposal dan skripsi saya sejak 2018,Staff Fotokopi Koperasi FMIPA UGM dan Fotokopi Perpustakaan Pusat UGM yang membantu kelengkapan berkas skripsi saya.

Akhir kata penulis berharap semoga skripsi ini dapat memberikan manfaat bagi kita semua, terutama bagi perkembangan ilmu pengetahuan serta perkembangan Ilmu Komputer dan Teknologi Informasi.

Yogyakarta, 18 Februari 2020

Mohamad Indra Wicaksono

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

INTISARI

PENDETEKSIAN API BERBASIS PENGOLAHAN CITRA DIGITAL MENGGUNAKAN METODE *CONVOLUTIONAL NEURAL NETWORK*

MOHAMAD INDRA WICAKSONO
13/348561/PA/15460

Sistem alarm api konvensional bergantung pada penggunaan sensor suhu atau asap. Performa dari system ini sudah akurat naumn bukan berarti tidak mempunyai kelemahan. Akurasi dari system ini bergantung pada jarak antar titik api dengan posisi sensor ditempatkan. Penelitian ini mengusulkan system berbasis pengolahan citra digital menggunakan convolutional neural network untuk melengkapi system alarm api yang sudah ada.

Convolutional neural network sebagai metode penelitian berbasis pengolahan citra digital sudah umum diterapkan, Namun penelitian yang mengaplikasikan *convolutional neural network* dalam bentuk *fully convolutional network* masihlah sangat sedikit. Adaptasi *Fully Convolutional Network 8 upsampled Prediction* (FCN-8s) ciptaan Long et al.(2015) digunakan sebagai metode yang diusulkan pada penelitian ini. Performa model yang diusulkan kemudian akan dibandingkan dengan pendeteksi api berbasis fully convolutional network yang diciptakan oleh Gonzales *et al.*(2017). Hasil menunjukkan bahwa performa FCN-8s mengalahkan performa SFEwAN-SD pada berbagai metrik baik dalam pengujian tingkat pixel dan tingkat citra.

Kata Kunci: *Convolutional Neural Network,* Pengolahan Citra Digital, Deteksi Api

ABSTRACT

DIGITAL IMAGE PROCESSING - BASED FIRE DETECTION USING CONVOLUTIONAL NEURAL NETWORK METHOD

MOHAMAD INDRA WICAKSONO
13/348561/PA/15460

Conventional fire alarm system depended on application of heat sensor and smoke sensor. Performance of said system is accurate however it isn't without flaw. Accuracy of most system depends on proximity of sensor against fire hotspot. This research proposed a system based on digital image processing using convolutional neural network method to complement existing fire alarm system.

Convolutional neural network as method in digital image processing research especially is already commonly used. However research that applies convolutional neural network in form of fully convolutional network still scarce. Adaptation of Fully Convolutional Network 8 upsampled Prediction (FCN-8s) by Long et al.(2015) is used as proposed method. Performance of proposed method also then compared against existing fully convolutional network-based fired detection by Gonales et al.(2017). Results shows FCN-8s performance beats SFEwAN-SD in various metrics both pixel level and image level test.

Keywords: Convolutional neural network, Digital Image Processing, Fire Detection

BAB I
PENDAHULUAN

1.1 Latar Belakang

Peristiwa kebakaran pada umumnya terjadi dari faktor yang tidak bisa kita prediksi. Peristiwa terjadinya kebakaran terjadi karena suatu material berada pada suhu yang tinggi sehingga api bisa terbentuk. Pada kebakaran hutan contohnya api bisa terjadi ketika daun kering atau batang kayu berada pada suhu yang sangat panas sehingga muncul api. Pada kasus dalam ruangan seperti di rumah atau pabrik titik api muncul bisa terjadi karena adanya arus pendek yang terjadi pada barang elektronik seperti setrika atau terminal stop kontak.

Sistem alarm api konvensional digunakan untuk memberikan peringatan akan adanya api dan mencegah api menyebar pada lebih lanjut.Pada umumnya untuk kondisi dalam ruangan untuk mendeteksi adanya api atau asap digunakan sensor pendeteksi asap atau sensor panas. Pada gudang misalnya, ruangan dilengkapi dengan sensor yang dilengkapi dengan alarm serta *sprinkler* untuk mencegah api menjalar lebih lanjut. Umunnya system alarm konvensional akurat namun akurasi system tersebut menurun seiring adanya batasan ruang yang dapat dijangkau oleh sensor tersebut, sehingga muncul perkembangan padanya system berbasis kecerdasan buatan sebagai untuk melengkapi kekurangan pada aspek luas ruang.

Kecerdasan buatan (*Artificial Intelligence*) mempunyai definisi umum perangkat lunak atau perangkat keras yang mampu berpikir dan belajar layaknya manusia yang mempunyai kecerdasan natural(*Natural Intelligence)*. Banyak penelitian yang melibatkan kecerdasan buatan karena penerapan dalam dunia nyata dirasa dapat menyelesaikan masalah yang tidak dapat diselesaikan dengan menerapkan metode ilmu komputer konvensional.

Penelitian pendeteksi api atau asap berbasis kecerdasan buatan muncul dalam berbagai pendekatan. Pendekatan yang dimaksud menggunakan karakteristik dari api seperti warna, bentuk, gerak, frekuensi, perubahan spasial dan asap yang

dihasilkan(Polednik, 2015). Beberapa diantaranya menggunakan sensor-sensor yang dapat mendeteksi asap dalam jarak dekat ataupun sensor thermal.

Computer Vision adalah bagian dari bidang kecerdasan buatan yang meniru bagaimana manusia menangkap informasi secara visual. Proses penangkapan informasi adalah sebagai berikut : Mendapatkan informasi dari mata, Mengetahui object yang dilihat dan terakhir meneruskan informasi dan mengolahnya. *Computer Vision* berusaha mendeskripsikan dunia yang manusia lihat dan merekontruksi karakteristiknya seperti bentuk, cahaya dan warna(Szeliski,2010). Proses itu didukung dengan integrasi model *machine learning* pada implementasinya. Peran *Computer Vision* dalam kehidupan nyata diperkirakan akan semakin bertambah dengan meningkatnya jumlah penelitian di bidang ini.

Perkembangan riset untuk deteksi api berbasis pengolahan citra digital bertujuan membantu mengenali adanya titik api lebih cepat di saat yang sama partikel asap dan api mengenai sensor yang dipasang. Penggunaan pada kehidupan sehari-hari pun lebih mudah dilakukan karena hanya perlu menggunakan CCTV(*Closed-Circuit Video television)* ataupun bentuk sistem video kamera lain seperti *drone*. Basis Penelitian menggunakan pengolahan citra digital menggunakan citra yang didapatkan dari hasil kamera/video yang kemudian diproses untuk mengenali objek yang direkam.

Penerapan Pengolahan Citra Digital pada CCTV umumnya digunakan untuk pengawasan seperti mencegah penyusup masuk dan mengawasi arus lalu lintas. Kedua contoh yang disebutkan adalah bentuk implementasi dari *object detection* dan *object tracking*. Algoritma yang sering digunakan untuk implementasi tersebut adalah algoritma *Camshift*. Algoritma *Camshift* pada dasarnya adalah penerapan *Mean-Shift* pada video. Algoritma *Mean-shift* pertama kali dikenalkan oleh Fukunaga *et al. (*1975). Mean-shift adalah pendekatan non-parametric yang bekerja dengan mencari modus dari sebuah *probability density function* dari data yang diberikan, dalam konteks ini data yang diberikan berupa *feature vector* dari citra. Secara umum, algoritma bekerja dengan mencari data point dengan densitas fitur yang terbesar, dalam hal ini contohnya adalah warna.

Mayoritas dari penelitan pendeteksian api berbasis pemrosesan citra digital lebih banyak berkutat pada ekstraksi fitur dan rekayasa fitur dari properti api. Proses rekayasa fitur tersebut terlalu memakan waktu dan memliki akurasi yang rendah. Metode konvensional juga seringkali salah dalam menafsirkan object pada object yang memiliki bayangan,pencahayaan yang variatif dan object dengan warna mirip dengan api.

Algoritma *Artificial Neural Network* khususnya *Convolutional Neural Network* menjadi trend dalam berapa tahun terakhir karena adanya minat yang tinggi pada bidang *Deep Learning*. *Convolutional Neural Network* menjadi sorotan karena kemampuannya dalam mengenali object. Beberapa tahun belakangan penelitian menggunakan metode *convolutional neural network* untuk mendeteksi api menunjukkan hasil akurasi yang tinggi seperti pada penelitian Frizzi *et al. (*2016), Gonzales *et al. (*2017) , Muhammad *et al. (*2017), dan Muhammad *et al. (*2018). Penelitian yang disebutkan menerapkan *convolutional neural network* biasa kecuali Gonzales *et al.* yang menerapkan *fully convolutional network*. *Fully convolutional network* adalah tipe *convolutional neural network* yang mampu memahami objek dalam tingkat segmentasi semantik atau dalam kata lain mampu memahami secara detil dengan objek dalam sebuah citra dengan mensegmentasi setiap objek dalam citra secara detil.

Fully convolutional network yang diterapkan oleh Gonzales *et al. (*2017) mempunyai basis arsitektur menggunakan AlexNet, arsitektur yang dibuat oleh Krhizevsky *et al. (*2012). Namun belum ada penelitian yang menggunakan basis arsitektur *state-of-the-art* lain seperti GoogLeNet atau VGGNet. Long et al.(2017) mengubah Alexnet,VGGNet dan GoogLeNet menjadi fully convolutional network yang kemudian didapatkan *fully convolutional network* dengan basis VGGnet mempunyai akurasi tertinggi dibandingkan dengan dua arsitektur lainnya. Arsitektur tersebut dinamakan *Fully Convolutional Network 8 upsampled Prediction* atau disingkat FCN-8s. Penelitian ini akan fokus pada implementasi *fully convolutional network* dengan arsitektur FCN-8s untuk mendeteksi api.

1.2 Rumusan Masalah

1. Apakah implementasi *fully convolutional network* dengan bentuk FCN-8s untuk pendeteksian api mempunyai akurasi yang lebih tinggi dibandingkan dengan penelitian berbasis *Fully Convolutional Network* yang sudah ada?
2. Apakah perubahan volume dataset untuk *training set* mengubah akurasi pada penelitian ?

1.3 Batasan Masalah

1. Penelitian hanya menerapkan kombinasi arsitektur dengan modifikasi *fully convolutional network* yang mempunyai persentase tertinggi untuk deteksi api.
2. Penelitian fokus pada deteksi api dengan cahaya api berwarna merah.
3. Dataset yang digunakan hanya berupa citra dengan adanya api berwarna merah.

1.4 Tujuan Penelitian

1. Melihat akurasi implementasi *fully convolutional network* pada deteksi api.
2. Mencari struktur yang optimal pada pendeteksian api.

1.5 Manfaat Penelitian

1. Memperlihatkan potensi *fully convolutional network* pada deteksi api.
2. Menghilangkan faktor jarak dan sensor panas pada deteksi api dengan menggunakan CCTV.
3. Diharapkan metode yang diusulkan dapat digunakan untuk sistem monitoring cerdas berbasis CCTV.

1.6 Metodologi Penelitian

Studi Literatur

Studi literatur dilakukan dengan pengumpulan literatur terkait dengan pendeteksian api berbasis pemrosesan citra digital dengan fokus pada metode convolutional neural network. Literatur yang digunakan dapat berupa jurnal, prosiding, buku atau karya ilmiah dari bidang yang bersangkutan. Studi juga meliput implementasi *fully convolutional network* yang relevan untuk digunakan pada penelitian ini.

Rancangan dan analisis

Tahap ini berupa melakukan proses penyesuaian rancangan masukan dan keluaran serta arsitektur yang akan dibuat. Arsitektur yang dibuat awalnya tidak dikhususkan untuk mendeteksi api sehingga diperlukan analisis dan rancangan lebih lanjut untuk diaplikasikan pada penelitian ini.

Implementasi

Implementasi dilakukan dengan menerapkan hasil rancangan dan analisis. Pada tahap ini akan dilakukan proses pelatihan arsitektur yang diusulkan beserta proses *fine-tuning* untuk mendapatkan hasil yang optimal. Kemudian

Evaluasi

Evalasi bertujuan melihat performa dari model yang diusulkan dengan menggunakan metrik evaluasi yang telah ditentukan. Hasil dari proses implementasi akan diuji dan dibandingkan dengan penelitian sebelumnya. Pada tahap ini akan dilihat kekurangan dan kelebihan dari arsitektur yang diusulkan.

1.7 Sistematika Penulisan

BAB I PENDAHULUAN

Bab I berisi latar belakang, rumusan masalah batasan masalah, tujuan penelitian, manfaat penelitian, metodologi penelitan dan sistematika penulisan.

BAB II TINJAUAN PUSTAKA

Bab II berisi tinjauan pustaka dari penelitian-penelitan terkait dengan tema penelitian ini. Tinjauan pustaka setiap penelitian terkait meliputi metode, dataset dan hasil. Bagian akhir tinjauan pustaka disertai dengan tabel perbandingan penelitian.

BAB III LANDASAN TEORI

Bab III berisi landasan teori yang terdiri dari teori dan definisi yang digunakan dalam penelitan, Teori dan definisi yang dituliskan dalam lingkup mengenai citra digital, *computer vision*, *machine learning*, *artificial neural network* dan *convolutional neural network*.

BAB IV ANALISIS DAN RANCANGAN

Bab IV berisi rancangan analisis dan rancangan pada dataset, arsitektur, pelatihan dan metode evaluasi yang akan digunakan pada penelitian.

BAB V IMPLEMENTASI

Bab V berisi implementasi pada rancangan yang dibahas pada Bab IV, meliputi perangkat penelitian, prapemrosesan dataset, implementasi *deep learning*, implementasi proses pelatihan, implementasi hasil prediksi, implementasi pengubahan lairk menjadi citra dan implementasi evaluasi.

BABVI HASIL DAN PEMBAHASAN

Bab VI berisi analisis dan pembahasan yang didapatkan setelah implementasi. Terdiri dari Hasil pembuatan ground truth, normalisasi dataset, hasil pelatihan dan perbandingan dengan arsitektur penelitian sebelumnya.

BAB VII KESIMPULAN

Bab VII berisi kesimpulan dari penelitian yang didapatkan dari hasil dan pembahasan serta saran untuk penelitian kedepannya.

BAB II
TINJAUAN PUSTAKA

Toreyin *et al. (*2005) menggunakan model *Hidden Markov* untuk mendeteksi api dari karakteristik temporal dan spasial . Pada penelitian ini moving region dideteksi terlebih dahulu. Setelah itu dilakukan pemeriksaan warna pada moving pixel, jika warna yang dideteksi sama dengan api maka akan diterapkan HMM pada karakteristik temporalnya kemudan diterapkan HMM secara spasial. Hasil dari eksperimen pada video masih menunjukkan adanya *false alarm* pada objek berwarna api yang bergerak.

Ko *et al. (*2011) Menggunakan *Fuzzy Finite Automata* (FFA) dengan *probability density function* pada fitur visual untuk mendeteksi karakteristik tidak regular pada api seperti bentuk yang berubah-rubah atau pola warna. Pendekatan ini dilakukan dengan mendeteksi wilayah yang bergerak pada video kemudian mencari kemungkinan wilayah api. Setelah itu *probability density function* pada fitur seperti intensitas, energi *wavelet,* dan orientasi gerak pada wilayah tersebut diaplikasikan pada FFA. Hasil dari pendekatan ini menunjukan FFA lebih mudah mendeteksi api pada video dengan tingkat pencahayaan rendah, adanya bayangan, benda menyerupai api dan gerak pada api.

Foggia *et al. (*2015) menggunakan pendekatan *multiexpert system* untuk deteksi api. Pada *multiexpert system* penentuan keputusan dilakukan dengan memecah feature vector dari input dan menerapkan berdasarkan tipe classifier yang sudah dispesifikasikan untuk satu fitur tertentu. Pada penelitian ini Foggia menggabungkan tiga macam expert system berdasarkan tiga fitur dari api yaitu : warna, variansi bentuk dan analisis gerak. Hasil dari penelitian ini mengalahkan akurasi dan efisiensi komputasi penelitian yang menggunakan *expert system* tunggal atau pendekatan non-*expert system* lainnya

Dimitropoulos *et al. (*2015) memilih pendekatan *spatio-temporal modelling* dan d*ynamic texture analysis* pada penelitiannya. Kedua teknik tersebut dikombinasikan karena pemodelan api menggunakan pendekatan spasial, temporal maupun percobaan mengkombinasikan keduanya dihadapi dengan masalah objek bukan api yang mempunyai fitur spasial atau temporal yang sama dengan api

sehingga tingkat *false alarm* tinggi. *Dynamic texture analisis* menurut Cetin *et al. (dalam* Dimitropoulos *et al.*,2015:1) mempunyai tingkat akurasi yang tinggi untuk klasifikasi video namun mempunyai tingkat komputasional yang tinggi dan metode yang ada tidak mendukung deteksi otomatis wilayah dari sebuah video sehingga tidak cocok untuk kebutuhan praktis seperti deteksi api.

Frizzi *et al. (*2016) melakukan penelitian dengan menerapkan *convolutional neural network* untuk Deteksi video api dan asap. Menurutnya penggunaan *rule-based model* dan *feature vector* tidak optimal. Dengan algoritma-algoritma convensional sulit mendefinisikan fitur yang ada dan akurasi dari deteksi bergantung pada besarnya api. Sehingga tingkat akurasinya rendah dan tingkat *false-alarm* yang tinggi. Pada penelitiannya CNN diimplementasikan karena akurasinya yang tinggi untuk klasifikasi objek. Pendekatan yang di lakukan adalah tiap layer berfungsi sebagai filter deteksi untuk pola atau fitur spesifik yang bisa secara mudah dikenali. Semakin dalam layer maka fitur yang di interpretasikan akan semakin abstrak. Pada layer yang terakhir CNN dapat menentukan klasifikasi spesifik yang menggabungkan semua fitur spesifik dari layer sebelumya. Hasil dari penelitian menunjukan tingkat akurasi 97.9 %.

Pendekatan *Fully Convolutional Network* digunakan Gonzales *et al. (*2017) untuk deteksi api pada kebakaran hutan menggunakan *Unmanned Aerial Vehicle (UAV)*. Pendekatan yang dilakukan menggunakan arsitektur yang didasari oleh AlexNet bernama *Simple Feature Extraction with FCN AlexNet, Single Deconvolution* (SFEwAN-SD). Arsitektur ini adalah integrasi dari dua model CNN. Model yang pertama adalah adaptasi alexnet untuk karakterisasi api berdasarkan bentuk dan tekstur. Model selanjutnya adalah sekuens convolutional untuk mengambil fitur tekstur dan warna. Gonzales et al juga membandingkan hasil akurasi untuk penelitiannya dengan model yang dibuat oleh Frizzy *et al. (*2016). Dari hasil eksperimen SFEwAN-SD mendapatkan akurasi sebesar 94.76 % dan mengalahkan model Frizzy yang mendapatkan akurasi hanya sebesar 86.96 %.

Penelitian Khan *et al. (*2017) mengusulkan teknik deteksi api dengan menggunakan fitur warna dan karakteristik *spatiotemporal* untuk mendeteksi wilayah api dengan bantuan *neural network*. Penelitian ini menggunakan

segmentasi warna dalam ruang warna *RGB* (*Red Green Blue*) kemudian perubahan dinamis dari fitur tersebut digunakan untuk menetapkan *foreground* yang bergerak. Menurut Khan, pendeteksian *foreground* dapat dilakukan dengan melihat perubahan variasi komponen warna biru pada ruang warna RGB karena pada rangkaian citra yang diuji nilai warna biru selalu variatif namun untuk merah dan hijau konstan.

Muhammad *et al. (*2017) menerapkan CNN dengan arsitektur yang mirip AlexNet, dengan modifikasi. Model yang dibuat mempunyai lima *convolutional layer*, tiga *pooling layer* dan tiga *fully connected layer*. Model yang digunakan tidak memerlukan pre-processing atau feature engineering karena model mempelajari fitur yang ada dari data secara langsung. Eksperimen dilakukan menggunakan dataset dari Foggia *et al.* (2015) dengan 14 video dengan api dan 17 video tanpa api. Video yang ada juga bervariasi dengan adanya objek berbentuk aapi seperti awan atau asap, atau berwarna api seperti matahari. Hasil dari eksperimen dengan akurasi 94.39 % mengalahkan model lain yang menggunakan pendekatan *machine learning* lainnya.

Tahun berikutnya Muhammad *et al. (*2018) melakukan penelitian menerapkan CNN untuk deteksi api menggunakan CCTV dengan arsitektur yang terinspirasi oleh GoogLeNet untuk mendapatkan hasil yang *cost-effective*. Arsitektur yang disebut dipakai karena secara komputasi tidak semahal AlexNet. GoogLeNet mempunyai akurasi klasifikasi yang lebih tinggi, model yang kecil dan cocok digunakan pada hardware dengan keterbatasan memori. Menurutnya mayoritas dari riset yang ada fokus ke metode ekstraksi fitur untuk deteksi api, masalahnya adalah metode tersebut memakan waktu yang lama dan performanya yang renda serta tingkat *false-alarm* yang tinggi. *False-alarm* yang dihasilkan karena adanya variasi terhadap object dengan bayangan, pencahayaan yang berbeda-beda, dan benda dengan warna yang mirip dengan api.

Hasil dari eksperimen setelah fine-tuning menunjukkan tingkat akurasi sebesar 94.43 % dengan false-positive sebesar 0.054 % dan false negative sebesar 1.5% mengalahkan pendekatan dari yang tahun sebelumnya dipublikasikan dengan dataset yang sama. Eksperimen juga dilakukan menggunakan dataset yang

digunakan oleh Chino et al. Dataset ini berbeda dengan dataset sebelumnya yang bervariatif dalam bentuk kontur dengan warna. Dalam dataset ini lebih banyak object dengan warna yang mirip dengan api atau pencahayaan dengan warna api. Namun hasil dari eksperimen hanya mendapatkan 86 %.

Tabel 2.1 Perbandingan Penelitian

No	Judul Penelitian	Penulis	Teknik
1	Flame detection in video using hidden Markov models	Toreyin *et al.*	Model Hidden Markov
2	Modeling and Formalization of Fuzzy Finite Automata for Detection of Irregular Fire Flames	Ko *et al.*	Fuzzy Finite Automata
3	Real-Time Fire Detection for Video-Surveillance Applications Using a Combination of Experts Based on Color, Shape, and Motion	Foggia *et al.*	Multiexpert system
4	Spatio-Temporal Flame Modeling and Dynamic Texture Analysis for Automatic Video-Based Fire Detection	Dimitropoulos *et al.*	Spatio-temporal modelling dan dynamic texture analysis
5	Convolutional neural network for video fire and smoke detection	Frizzi *et al.*	Convolutional Neural Network
6	Accurate Fire Detection through Fully Convolutional Network	Gonzales *et al.*	FCN berdasarkan AlexNet
7	Real-Time Fire Detection Using Enhanced Color Segmentation and Novel Foreground Extraction	Khan *et al.*	Segmentasi warna dengan bantuan Neural Network
8	Early fire detection using convolutional neural networks during surveillance for effective disaster management	Muhammad *et al.*	CNN dengan arsitektur berdasarkan AlexNet
9	Convolutional Neural Networks Based Fire Detection in Surveillance Videos	Muhammad *et al.*	CNN dengan Arsitektur berdasarkan GoogLeNet

BAB III
DASAR TEORI

3.1 Citra Digital

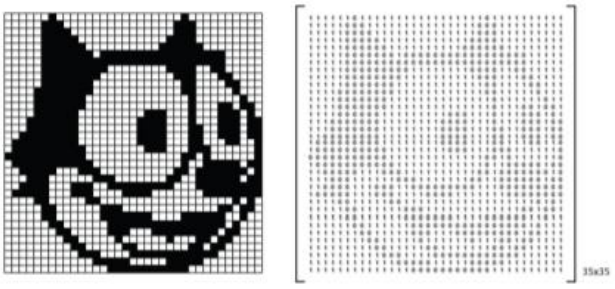
Perkembangan citra digital pada awalnya dikembangkan sebagai solusi medium pengiriman citra dari London menuju New York di awal tahun 1920 untuk industry koran. Sistem ini disebut *Bartlane cable picture transmission system.*Sistem ini bekerja dengan menggunakan peralatan pencetakan khusus yang menerima data citra yang sudah *encoded(*dirubah menjadi bentuk kode) menjadi sebuah citra yang dicetak pada plat khusus. Seiring perkembangan teknologi citra digital mulai menggantikan citra analog sebagai medium gambar.

3.1.1 *Definisi Citra Digital*

Citra digital dapat didefinisikan sebagai matrix dua dimensi di mana setiap element pada matrix mempunyai nilai tertentu yang disebut *picture element* atau dikenal sebagai *pixel.* Nilai dari sebuah *pixel* pada citra menunjukkan nilai diskrit yang merepresentasikan intensitas warna. Secara umum representasi citra digital dibagi menjadi tiga jenis berdasarkan nilai dari *pixel*-nya yaitu citra biner, citra kelabu dan citra warna.

3.1.2 *Representasi Citra*

Citra Biner

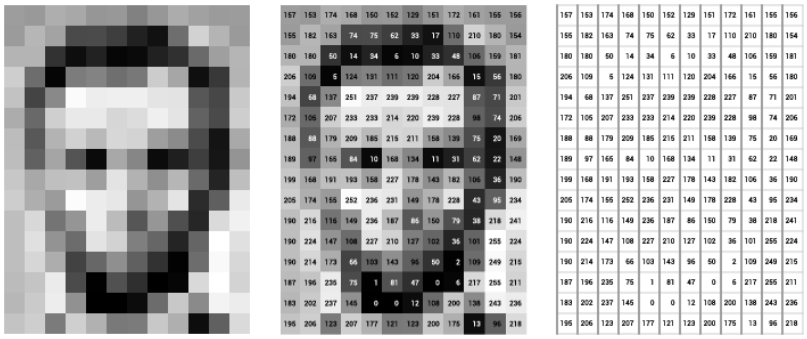


Sumber : http://www.uff.br/cdme/matrix/matrix-html/matrix_boolean/matrix_boolean_en.html

Gambar 3.1 Citra kelabu

Citra biner adalah citra digital dengan nilai pixel berupa 0 atau 1 dimana nilai yang dialokasikan pada *pixel* adalah hitam ataupun putih. Citra biner pada gambar 3.1 adalah sebuah matrix dua dimensi berukuran 35x35 dengan 0 dialokasikan untuk warna putih dan 1 untuk warna hitam.

Citra kelabu(*grayscale)*

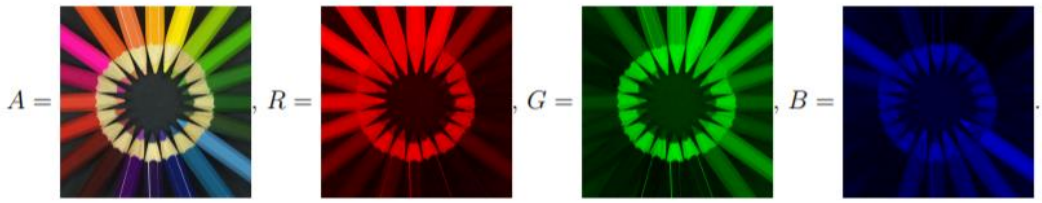


Sumber : http://gameautomators.com/book/ImageProcessing/2-color.html

Gambar 3.2 Citra Kelabu

Citra kelabu adalah citra yang nilai elemennya merepresentasikan dengan intensitas warna dari citra tersebut dengan jangkauan warna dari putih sampai dengan hitam. Jangkauan tersebut biasanya mempunyai nilai 0-255 yang disimpan pada memori sebesar 8 bit.

Citra Warna



Sumber : http://www.uff.br/cdme/matrix/matrix-html/matrix_color/matrix_color_en.html

Gambar 3.3 Citra Warna

Citra warna pada dasarnya sama dengan citra kelabu di mana pixel merepresentasikan intensitas warna pada citra. Perbedaannya adalah pada citra warna suatu pixel tidak hanya merepresentasikan lebih dari satu warna, tergantung *color model* yang digunakan. Pada gambar 3 citra warna didefinisikan pada *color model* RGB (*Red Green Blue*). Satu pixel pada citra warna merupakan kombinasi dari tiga bagian dari *color model* tersebut, *Color model* yang sering digunakan pada citra warna selain RGB adalah HSI(*Hue Saturation Intensity*).

3.1.3 *Video Digital*

Video digital adalah representasi citra bergerak yang yang ditampilkan secara sekuensial. Citra bergerak yang ditampilkan adalah citra digital yang mempunyai format tertentu. 1 citra pada video disebut *frame*. Pada video digital terdapat standard kecepatan jumlah frame yang ditampilkan dalam 1 detik yang

disebut *frame per second(FPS)*. Semakin banyak jumlah frame yang ditampilkan dalam 1 detik berbanding lurus dengan kehalusan gerak pada video. Biasanya tingkat FPS pada video digital berkisar antara 24 – 30 fps.

3.2 Computer Vision

Computer vision adalah bidang ilmu yang menjadi sorotan pada kecerdasan buatan khususnya pengenalan objek. Pada dasarnya tujuan *computer vision* adalah mencoba meniru bagaimana manusia mengenali dan menganalisis objek melalui proses pengolahan citra digital. Beberapa tahun belakangan perkembangan pesat *convolutional neural network* menjadi alasan bidang ilmu ini makin popular.

3.2.1 *Definisi Computer Vision*

Computer Vision adalah konstruksi eksplisit dengan deskripsi bermakna sebuah objek fisik dari kumpulan citra. Memahami sebuah citra berbeda dengan memprosesnya, di mana pemprosesan citra adalah transformasi dari satu bentuk citra ke bentuk citra lain, bukan pemaknaan dari deskripsi citra. Deskripsi adalah prasyarat untuk mengenali, memanipulasi dan memikirkan objek.(Ballard & Brown, 1982)

Menurut T.S. Huang, pionir dari bidang *Computer Vision* berasal dari tesis Ph.D oleh Lawrence Gilman Roberts di MIT (*Massachusetts Institute of Technology*) yang berjudul *"Machine Perception of Three Dimensional Solids"*. Tesis tersebut mendiskusikan kemungkinan mengekstraksi informasi geometri tiga dimensi dari perspektif dua dimensi dari polyhedra. Setelah itu pada 1978 David Marr pada bukunya yang berjudul *"Vision : A Computational Investigation into the Human Representation and Processing of Visual Information"* mengusulkan paradigma baru untuk mempelajari Visi. Menurutnya visi bisa dideskripsikan menjadi tiga tingkatan yaitu :

1.*computational theory*

apakah tujuan dari komputasi, kenapa itu pantas dan apa logika dari strategi yang dapat digunakan untuk mencapainya ?

2.*representation and algorithm*

Bagaiman teori komputasi itu bisa diimplementaikan ? Khususnya apakah representasi untuk masukan dan keluaran serta algoritma untuk transformasinya?

3.*hardware implementation*

Bagaimana representasi dan algoritma bisa direalisasikan secara fisik ?

Computer vision merupakan bidang yang sulit karena masalah yang berusaha diselesaikan adalah *inverse problem*. Dalam *Inverse problem* solusi yang dicari didapatkan dari sumber informasi yang tidak mencukupi. Dalam konteks *Computer Vision* berarti solusi yang dicari dari sebuah citra adalah merekonstruksi properti dari citra tersebut seperti bentuk, pencahayaan, distribusi dan distribusi warna. (Szeliski, 2010).

3.3 Machine Learning

Machine learning atau pembelajaran mesin adalah bidang ilmu yang merupakan salah satu fokus dari bidang kecerdasan buatan. Bidang ini berbeda dengan bidang di dalam kecerdasan buatan lainnya karena tidak memerlukan programming secara eksplisit dalam pengembangannya, namun menggunakan sample data atau dataset sebagai acuan untuk membentuka keputusan.

3.3.1 *Definisi Machine Learning*

Machine Learning adalah seperangkat metode yang bisa mendeteksi pola dari data secara otomatis dan menggunakan pola yang didapat untuk memprediksi data di masa yang akan datang atau melakukan bentuk pengambilan keputusan lain dalam ketidakpastian (Bishop, 2006). Berdasarkan metodenya, bentuk dari Machine learning itu sendiri dapat dibagi menjadi tiga tipe berdasarkan feedbacknya, yaitu *Supervised Learning*, *Unsupervised Learning* dan *Reinforcement Learning*. Penelitian ini menerapkan *supervised learning* untuk mengklasifikasikan video dengan api dan tanpa api.

3.3.2 *Supervised Learning*

Supervised learning adalah tipe pembelajaran mesin di mana goal dari sebuah agen kecerdasan buatan memperkirakan output berdasarkan contoh pasangan input-ouput. Pasangan input-ouput tersebut disebut *training data* atau *training set*. Pada *training set* data input berupa vektor dengan nilai-nilai yang

merepresentasikan fitur dalam bentuk angka. Bentuk dari output adalah *response variable* dengan tipe data kategorikal untuk masalah klasifikasi atau nominal untuk masalah regresi.

Klasifikasi

Klasifikasi adalah kasus dimana output yang diinginkan dari supervised learning berupa data kategori yang diskrit. Salah satu contoh dari masalah klasifikasi adalah pengenalan digit tulisan tangan. Dalam masalah tersebut, computer harus mengenali digit dari 0-9 dari berbagai bentuk tulisan tangan. Contoh lainnya adalah implementasi *spam filter* untuk memisahkan antara email asli dengan *spam*.

Regresi

Regresi adalah tipe masalah dimana output yang diinginkan berupa data nominal atau kontinyu. Contoh regresi seperti memprediksi harga saham di masa yang akan datang jika diberikan harga saham terkini atau parameter lainnya.

3.4 Artificial Neural Network

Artificial Neural network atau jaringan saraf tiruan merupakan salah satu metode pertama dan tertua yang dikembangan dalam bidang *machine learning*. Perkembangan *neural network* pada tahun 2010 dan setelahnya mendapatkan perhatian besar karena memenangkan berbagai kompetisi pengenalan objek. Salah satu yang menjadi sorotan adalah AlexNet yang memenangkan Large Scale Visual Recognition Challenge 2012 dengan tingkat galat 15,8%, peningkatan signifikan dari tahun-tahun sebelumnya dengan tingkat galat rata-rata di atas 20%.

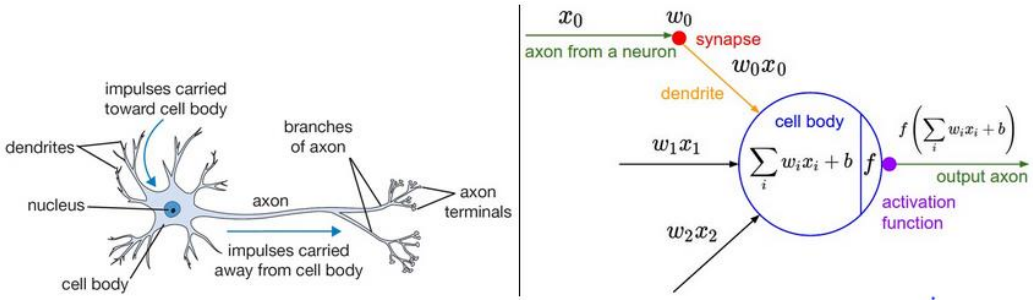
3.4.1 *Definisi Artificial Neural Network*

Inspirasi dari *Artificial Neural Network* adalah system pemrosesan informasi yang mempunyai karakteristik performa yang sama dengan jaringan saraf biologis. *Artificial Neural Network* dikembangkan sbagai generalisasi untuk model matematis dari kemampuan kognitif manusia atau biologi syaraf berdasarkan asumpsi sebagai berikut :

1. Pemrosesan infrormasi terjadi di elemen sederhana bernama neuron.
2. Sinyal diteruskan antara neuron melalui jaringan terhubung.

3. Tiap jaringan terhubung mempunyai *weight* yang diasosiasikan dimana jika sinyal melewati jaringan tersebut maka sinyal akan dikalikan dengan weight tersebut.

4. Setiap neuron menerapkan fungsi aktifasi(umumnya non-linear) kepada *net input(*jumlah sinyal *input* yang sudah dikalikan weight).(Fausett ,1994)



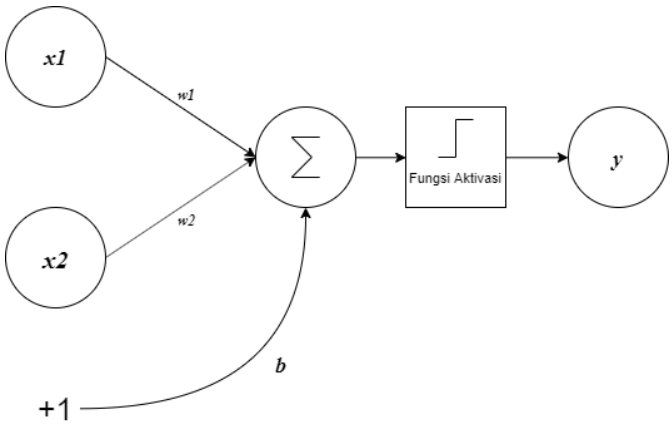
Sumber : http://cs231n.github.io/convolutional-networks/

Gambar 3.4 Perbandingan *Biological* dan *Artificial Neural Network*

3.4.2 *Arsitektur Neural Network*

Arsitektur pada neural network berupa neuron yang disusun menyerupai suatu layer atau lapisan. Setiap layer mempunyai fungsi aktifasi yang sama dan terhubung dengan setiap neuron pada layer berikutnya. Bentuk dari arsitektur neural network dibagi menjadi dua, *feedforward neural network* dan *recurrent neural network.* Pada *feedforward neural network* setiap neuron asklik, yang berarti informasi berjalan satu arah dari input ke output. Contoh paling sederhana dari jenis neural network ini adalah *Perceptron*.

Perceptron



Gambar *3.5 Perceptron*

Perceptron adalah jenis *neural network* yang dikembangkan oleh Frank Rosenblatt di tahun 1958. Pada dasarnya Perceptron menjumlahkan sejumlah input *x* yang dikalikan dengan bobot *w* serta *bias* untuk menghasilkan suatu keluaran. Hasil penjumlahan tersebut kemudian dikenakan fungsi aktivasi, yang bertugas menentukan apakah keluaran tersebut dapat diteruskan ke neuron selanjutnya.

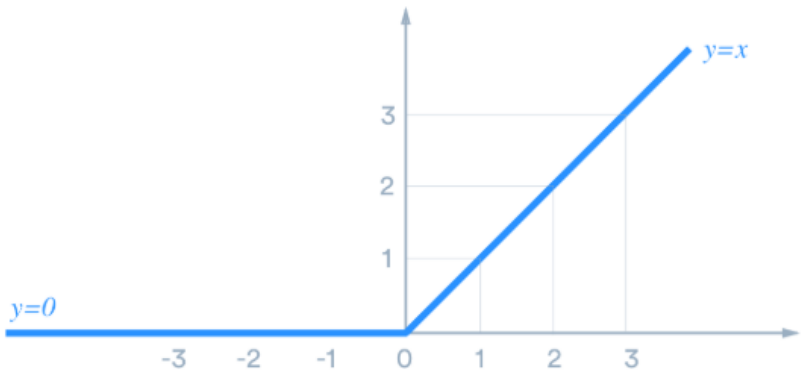
3.4.3 *Fungsi Aktivasi*

Relu

Rectified Linear Unit atau ReLU dikenalkan Hahnloser et al. adalah fungsi aktivasi yang popular digunakan untuk implementasi neural network. Kelebihan dari ReLU adalah bentuknya yang sederhana. Pada persamaan 3.1 ReLU diformulasikan sebagai berikut:

$$y = max(0, x) \qquad 3.1$$

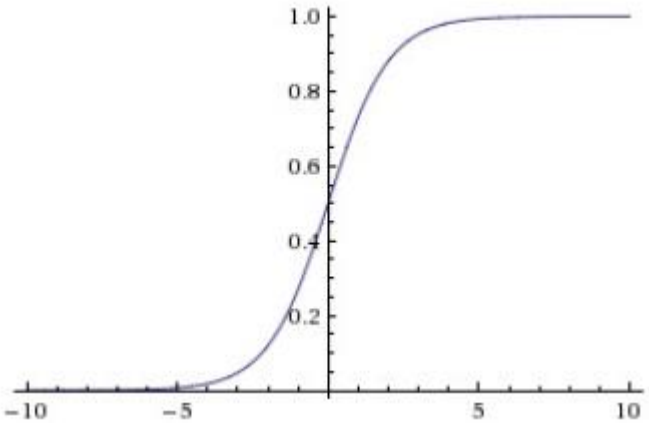
Dari persamaan tersebut kita bisa lihat bahwa fungsi ReLU tidak memerlukan komputasi yang kompleks, hanya menggunakan *max*(). Semakin sederhana komputasinya maka biaya komputasi implementasinya semakin murah.Selain itu karena nilai dari fungsi ini tidak mungkin negative maka hal ini menghindari terjadinya *vanishing gradient problem.*



Sumber: https://medium.com/@danqing/a-practical-guide-to-relu-b83ca804f1f7

Gambar 3.6 Fungsi ReLU

Sigmoid



Sumber: http://cs231n.github.io/neural-networks-1/

Gambar 3.7 Kurva Sigmoid

Fungsi sigmoid adalah fungsi aktivasi non-linear dengan hasil fungsi berupa bilangan riil antara 0 dan 1. Nama sigmoid berasal dari bentuk grafik yang dihasilkan (lengkung *sigmoid*). Fungsi logistic sigmoid didefinisikan sebagai persamaan berikut:

$$f(x) = \frac{1}{1 + e^{-x}} \qquad 3.2$$

Gambar 3.7 memperlihatkan bahwa untuk hasil fungsi untuk x yang mendekati nilai negatif akan menjadi 0 dan untuk x positif sebesar apapun x akan maksimum menjadi nilai 1.

3.4.4 *Loss Function*

Performa dari sebuah model *machine learning* dapat dilihat dari nilai *cost function*. Cost function atau Loss function adalah formula untuk melihat kemampuan model dalam memprediksi ketepatan relasi antara X dan *y*. Akurasi dari sebuah model dapat dilihat dari nilai cost function yang makin turun. Terdapat berbagai jenis cost function untuk kebutuhan yang spesifik. Dua diantarnya yang banyak digunakan adalah *Cross-entropy* dan *Hinge Loss*.

Cross-entropy

Cross-entropy loss adalah *loss function* untuk mengukur performa model klasifikasi dengan nilai probabiliatas antara 0 sampai 1. *Loss* diukur dengan

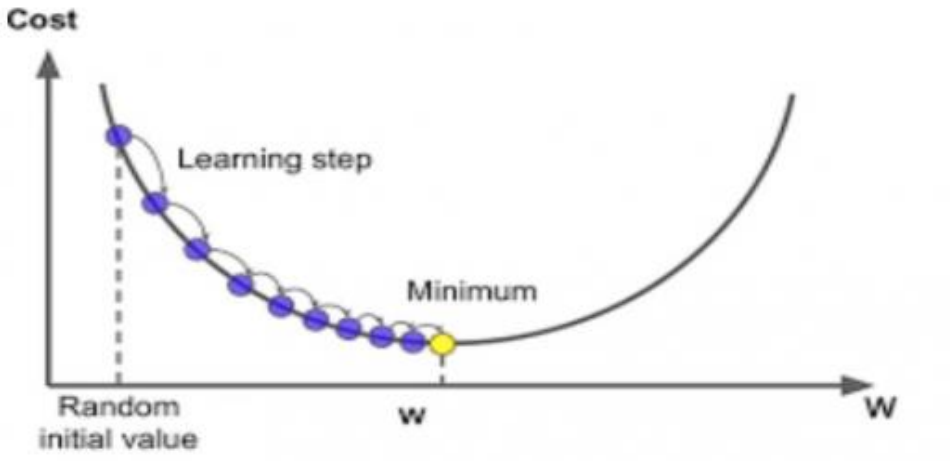
mengukur seberapa jauh nilai prediksi (*p)* dengan label asli *(y)*. Persamaan *crossentropy* untuk klasifikasi biner dapat didefinisikan sebagai berikut:

$$L = -(ylog(p) + (1-y)\,log(1-p)) \qquad 3.3$$

3.4.5 *Algoritma Optimasi*

Algoritma optimasi adalah bentuk optimasi matematis untuk meminimalisir *loss* dalam sebuah model matematis. Pada konteks *machine learning,* algoritma optimasi digunakan untuk mengubah weight berdasarkan loss yang ada agar meminimalisir loss di proses pelatihan berikutnya. Pada implementasinya algoritma optimasi yang paling sering digunakan adalah optimasi berbasis penurunan gradien (*gradient descent)*.

Gradient Descent



Sumber: https://www.topcoder.com/gradient-descent-in-machine-learning/

Gambar 3.8 Illustrasi penurunan gradien

Adam

Adam dikembangkan oleh Kingma dan Ba (2015) sebagai algoritma optimisasi berbasis gradient descent. Nama *Adam* berasal dari perpaduan *adaptive moment estimation* yang berasal dari cara Adam menggunakan estimasi momen pertama dan kedua dari gradient untuk mengadaptasi learning rate dari setiap *weight*. Metode ini bisa dikatakan kombinasi antara SGD dan RMSProp karena mempunyai kelebihan dari keduanya. Algoritma *Adam* dimulai dengan menghitung estimasi moment pertama(m_t) dan estimasi momen kedua (v_t) menggunakan

gradient dari fungsi objektif (g_t) serta konstanta *exponential decay rates* (β_1 dan β_2) dengan persamaan sebagai berikut:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t \quad 3.4$$

$$v_t = \beta_2 m_{t-1} + (1 - \beta_2) g_t^2 \quad 3.5$$

Setelah itu karena m_t dan v_t cenderung bernilai menuju 0 maka untuk mencegah hal itu terjadi langkah selanjutnya dilanjutkan dengan mengkoreksi hal terseut dengan persamaan berikut:

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t} \quad 3.6$$

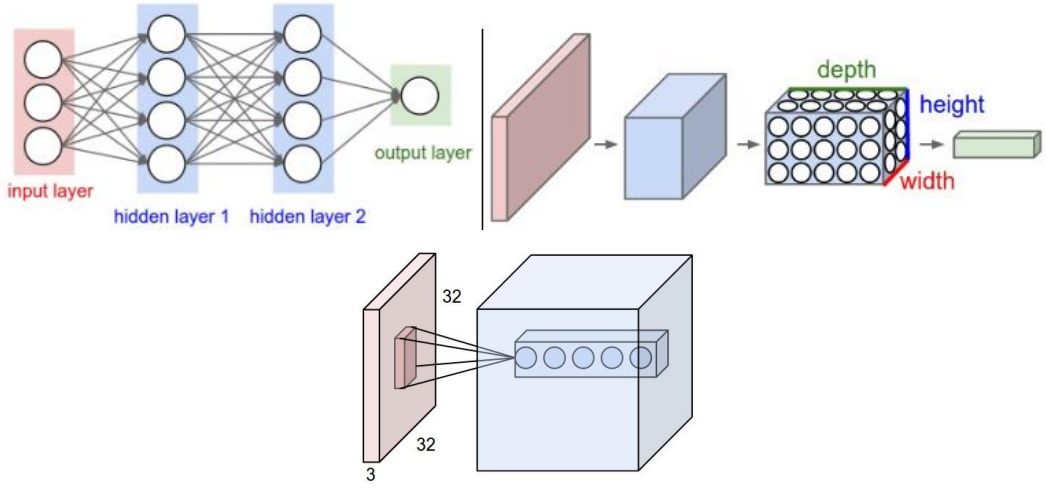
$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t} \quad 3.7$$

Langkah terakhir untuk mengupdate parameter dari fungsi objektif dengan variable *stepsizes(α)* dan *epsilon*(ϵ):

$$\theta_t = \theta_{t-1} - \frac{\alpha}{\sqrt{\hat{v}_t} - \epsilon} \hat{m}_t \quad 3.8$$

3.4.6 *Convolutional Neural Network*

Convolutional neural network atau CNN adalah neural network yang khusus untuk memproses data yang mempunyai topologi dengan bentuk *grid*. *Grid* adalah bentuk topologi di mana tiap *node* mempunyai koneksi dengan dua node yang menjadi tetangganya. Contohnya adalah seperti data *time-series* yang bisa dikatakan sebagai grid 1-D atau citra yang bisa dikatakan sebagai grid 2D (Goodfellow *et al.*, 2016). *Convolutional Neural Network* pertama kali dikenalkan pada oleh Yann LeCun pada 1989. Pada publikasinya CNN secara spesifik didesain untuk menghadapi variansi pada bentuk dua dimensi, dalam kasus ini pengenalan digit dalam tulisan tangan. *Convolutional neural network* berbeda dengan ANN karena menggunakan operasi matematika berupa *convolution* pada minimal satu *layer* nya sebagai ganti dari matriks multiplikasi. Berapa tahun belakangan convolutional neural network menjadi sorotan karena kemampuannya pada *object detection*.

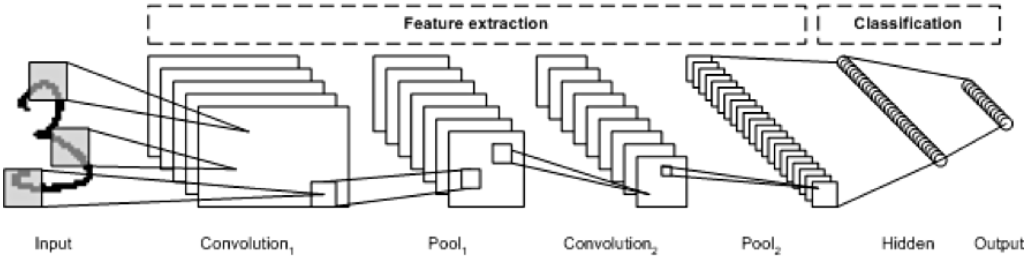


Sumber : http://cs231n.github.io/convolutional-networks/

Gambar 3.9 Perbandingan Artificial Neural Network biasa dengan Convolutional Neural Network

CNN bisa dikatakan sebagai arsitektur *ANN* yang awalnya dispesialisasikan untuk data berbentuk citra walaupun pada perkembangannya dapat diaplikasikan ke tipe data lain. Pada Gambar 5 di sebelah kiri dapat dilihat arsitektur ANN secara umum. Pada arsitektur ini setiap Neuron terhubung dengan semua neuron pada *layer* berikutnya. Karena 1 neuron menangkap data citra secara keseluruhan, maka *weight* dari neuron akan sebesar volume citra. Hal ini akan menimbulkan biaya komputasi yang terlalu mahal. *CNN* menghindari hal itu dengan melihat citra sebagai data 3 dimensi (panjang, lebar, dan kedalaman). Pada CNN, 1 neuron tidak melihat data citra secara keseluruhan namun hanya sebagaian kecil segmen pada citra yang disebut *receptive field.*

Pada umumnya setiap arsitektur CNN mempunyai tiga jenis layer utama yaitu *convolutional layer, pooling layer, dan fully-connected layer*. Gambar berikut merupakan contoh arsitektur *convolutional neural network* yang dibuat oleh Lecun *et al.* (1989) untuk mengenali digit 0 sampai 9 untuk tulisan tangan. Arsitektur ini terdiri dari 2 *convolutional layer* (*Convolution*$_1$ dan *Convolution*$_2$), 2 *pooling layer* (*Pool*$_1$ dan *Pool*$_2$), dan 1 *fully-connected layer* (*Hidden).*



Sumber : https://www.ibm.com/developerworks/library/cc-machine-learning-deep-learning-architectures/index.html

Gambar 3.10. Arsitektur *LeNet*

Convolutional Layer

Pada gambar 3.10 *convolutional layer* berada pada bagian Convolution_1 dan Convolution_2.Tujuan utama dari *convolutional layer* adalah mengekstraksi fitur dari masukan berupa citra menjadi *feature map* menggunakan *feature detector*. Pada layer ini fitur diekstrak dengan mengambil sebagian segmen pada citra atau dapat disebut *receptive field* menggunakan *feature detector*. Proses ekstraksi fitur ini dilakukan dengan operasi convolusi. Operasi convolutional dilakukan dengan hasil dot product antara citra dengan *feature detector*. Feature detector pada layer ini disebut filter. Filter ini berukuran lebih kecil daripada citra namun memindai semua bagian citra secara keseluruhan.

Di dalam *convolutional layer* setelah *feature map* terbentuk maka hasil konvolusi tersebut dimasukan dalam fungsi aktivasi. Fungsi aktivasi mempunyai tujuan meneruskan output menuju layer berikutnya ketika nilai dari input melewati titik ambang dari fungsi tersebut. Pada contoh kali ini fungsi aktivasi yang dipakai adalah *Rectified Linear Unit (ReLU)* dengan bentuk umum sebagai berikut :

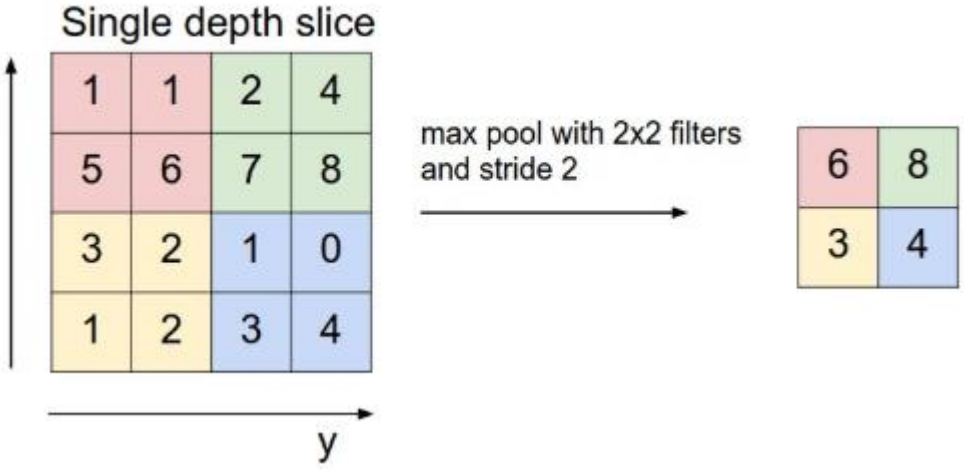
$$f(x) = max(0, x) \qquad 3.9$$

Pada Fungsi ReLU mempunyai batas titik ambang dengan nilai 0. Ketika nilai dari fungsi tersebut negative maka keluaran dari fungsi ini adalah 0. Dengan demikian *feature map* tersebut tidak diteruskan ke *layer* selanjutnya. Fungsi ini bertujuan mengeliminasi nilai negative pada *feature map* sehingga berguna mengurangi jumlah neuron yang diaktifkan dalam layer sehingga mengurangi biaya komputasi.

Pooling Layer

Arsitektur pada CNN biasanya berupa *convolutional layer* yang diikuti dengan *pooling layer*. Pada gambar di atas *pooling layer* ditunjukkan pada bagian

$Pool_1$ dan $Pool_2$ Secara intuitif pooling layer berfungsi mengurangi ukuran untuk mengurangi jumlah parameter dan komputasi yang ada di jaringan serta mencegah terjadinya *overfitting*. *Pooling* dilakukan dengan mengubah dimensi dari setiap *feature map* menjadi lebih kecil namun tetap menyimpan informasi penting yang diperlukan menggunakan *filter* dan *stride* dengan ukuran tertentu. Terdapat beberapa jenis dari operasi *pooling* diantaranya *max*, *average*, atau *L2-norm*.



Sumber : http://cs231n.github.io/convolutional-networks/

Gambar 3.11 Operasi *Max Pooling*

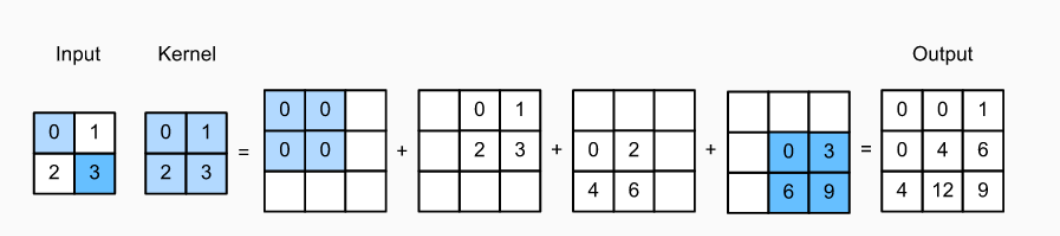
Gambar di atas merupakan contoh operasi *max pooling*. *Feature map* pada sisi sebelah kiri adalah feature map hasil dari *convolutional layer* dengan ukuran 4x4. Operasi *max pooling* di atas mempunyai *filter* sebesar 2x2 dan *stride* 2 pixel. Luas dari tiap filter ditunjukkan dengan warna yang berbeda (merah, hijau, kuning dan biru). Pada tiap bagian filter yang diambil hanya nilai paling tinggi, sehingga hasil dari operasi tersebut menghasilkan *feature map* yang mempunyai dimensi lebih kecil yaitu 2x2.

Fully Connected Layer

Fungsi utama pada layer ini adalah melakukan operasi klasifikasi pada input yang sudah melalui proses *convolutional* dan *pooling*. Pada gambar 3.10 *fully connected layer* ditunjukkan pada bagian *hidden*. Perbedaannya dengan jenis *layer* sebelumnya adalah pada *fully connected layer* semua neuron terhubung dengan semua neuron pada *layer* sebelumnya sama seperti *multi-layer perceptron* pada

neural network biasa. Klasifikasi pada layer ini dilakukan menggunakan *loss function* seperti *Support Vector Machine* atau *Softmax*.

Transpose Convolutional Layer

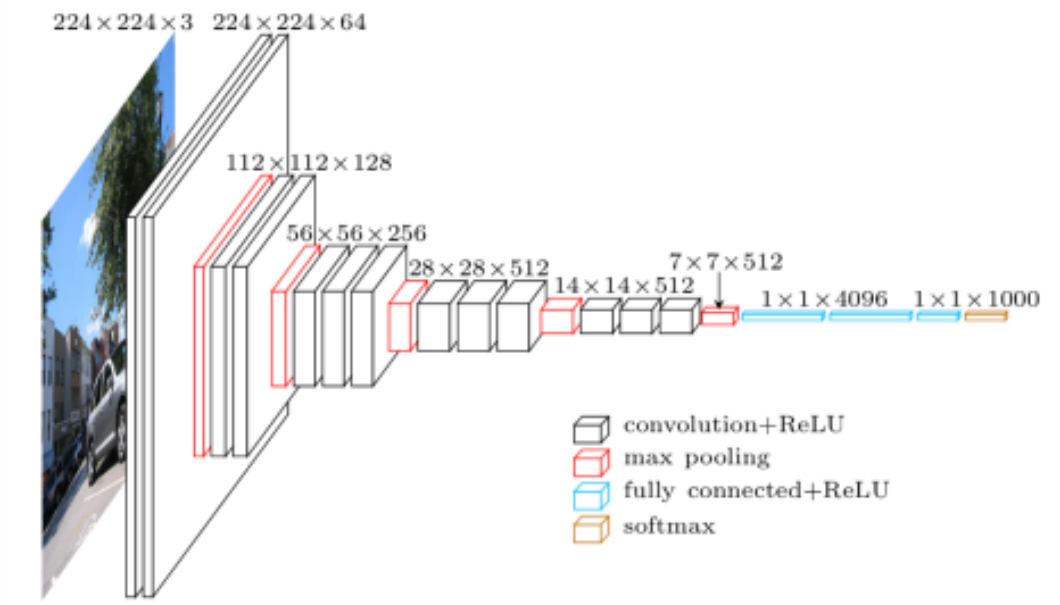


Sumber: https://d2l.ai/chapter_computer-vision/tranposed-conv.html

Gambar 3.12 Illustrasi operasi *transposed convolution*

Transpose convolutional layer adalah layer yang melakukan operasi konvolusi secara terbalik. Jika pada convolutional layer larik 2D direduksi menjadi sebuah feature map dengan dimensi yang berkurang maka transpose convolutional layer mengubah feature map menjadi bentuk larik 2D dengan dimensi yang diperbesar. Operasi yang terjadi pada layer ini disebut juga *fractionally strided convolution*. Operasi *transposed convolution* lebih popular dikenal dengan nama *deconvolution* walaupun secara terminologi tidak akurat.

3.4.7 *VGGNet*

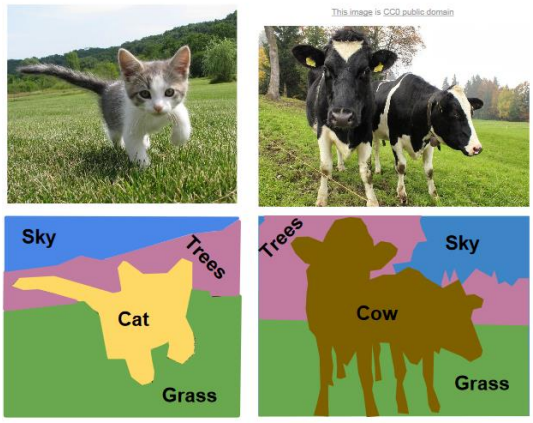


Sumber : https://www.cs.toronto.edu/~frossard/post/vgg16/

Gambar 3.13 Arsitektur VGGNet dengan 16 *layer*

VGGnet merupakan *runner-up* dari ILSVRC 2014 yang dibuat oleh Simonyan-Zisserman (2014). Arsitektur ini mempunyai beberapa variasi namun yang VGG dengan 16 dan 19 layer yang sering digunakan. Arsitektur ini unik karena hanya menggunakan 1 ukuran filter yaitu 3x3 pada implementasinya. Walaupun lebih sederhana dibandingkan dengan arsitektur *state-of-the-art* lainnya arsitektur ini mempunyai kelemahan pada biaya komputasi yang mahal dan waktu *training* yang lama. Pada 2015 Long *et al.* mengubah VGGNet menjadi *fully convolutional network* yang performanya melebihi *GoogLeNet* dan *AlexNet* pada segmentasi semantic.

3.4.8 *Fully Convolutional Network*

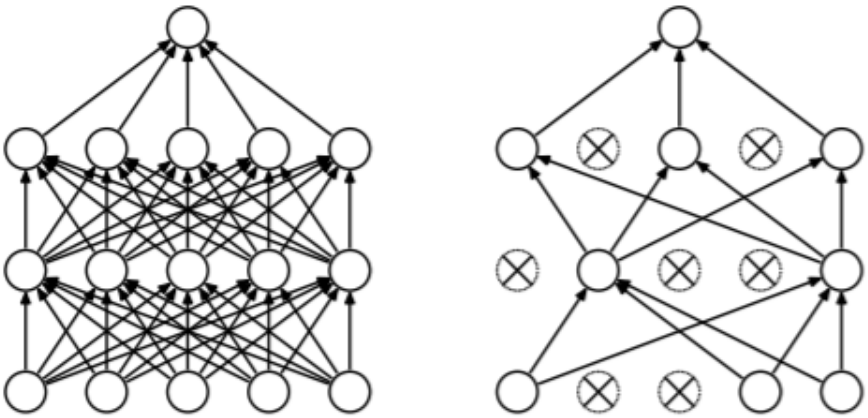


Sumber : https://www.analyticsvidhya.com/blog/2017/11/heart-sound-segmentation-deep-learning/

Gambar 3.14 *Contoh Segmentasi Semantik*

Istilah *fully convolutional network* dikenalkan oleh Long *et al. (*2015) yang fokus pada segmentasi semantik. Pengertian segmentasi semantik adalah memahami citra dalam tingkat pixel dimana setiap *pixel* mempunyai label yang berkorespondensi dalam objek atau wilayah yang bersangkutan.Tujuan dari implementasi FCN pada *paper*-nya adalah membuat arsitektur yang mampu menerima input dengan ukuran yang bervariasi serta output yang mempunyai ukuran yang sama dengan input namun dengan *learning* dan inferensi yang efektif. Long *et al.* mengubah tiga arsitektur CNN juara ISLVRC yaitu : AlexNet, GoogLeNet dan VGGnet yang kemudian diadaptasi menjadi FCN. Perubahan dilakukan dengan membuang *classifier layer* pada jaringan dan mengubah *fully connected layer* menjadi *convolutional layer.*

3.5 Dropout



Sumber : Srivastava *et al.*(2014)

Gambar 3.15 Illustrasi Dropout

Dropout dikembangkan oleh Srivastava *et al.*(2014) untuk mencegah overfitting pada model neural network. Semakin besar dan dalam sebuah model neural network maka akan semakin tinggi kemungkinan terjadi overfitting. Hal inin dikarenaka neural network mampu mengenali dan mempelajari fitur-fitur yang kompleks dan non-linear. Hal yang umumnya dilakukan untuk mengatasi overfitting adalah dengan mencoba kombinasi parameter dan arsitektur neural network untuk mencapai hasil yang optimal. Namun untuk melakukan berbagai kombinasi tersebut dibutuhkan kemampuan komputasi yang mahal dan waktu yang tidak sedikit. Dropout mengatasi masalah tersebut dengan membuang neuron secara acak.

BAB IV
ANALISIS DAN RANCANGAN

4.1 Analisis Dataset

Penelitian ini membutuhkan dataset yang terdiri dari 2 jenis citra, citra warna dan *ground truth* yang berupa citra kelabu dengan objek api. Hasil pencarian memperlihatkan bahwa ketersediaan dataset yang mempunyai *ground truth* tidaklah mudah didapatkan. Dari hasil pencarian didapatkan hanya 1 dataset yaitu *Bowfire* dataset yang disertai oleh *ground truth*.

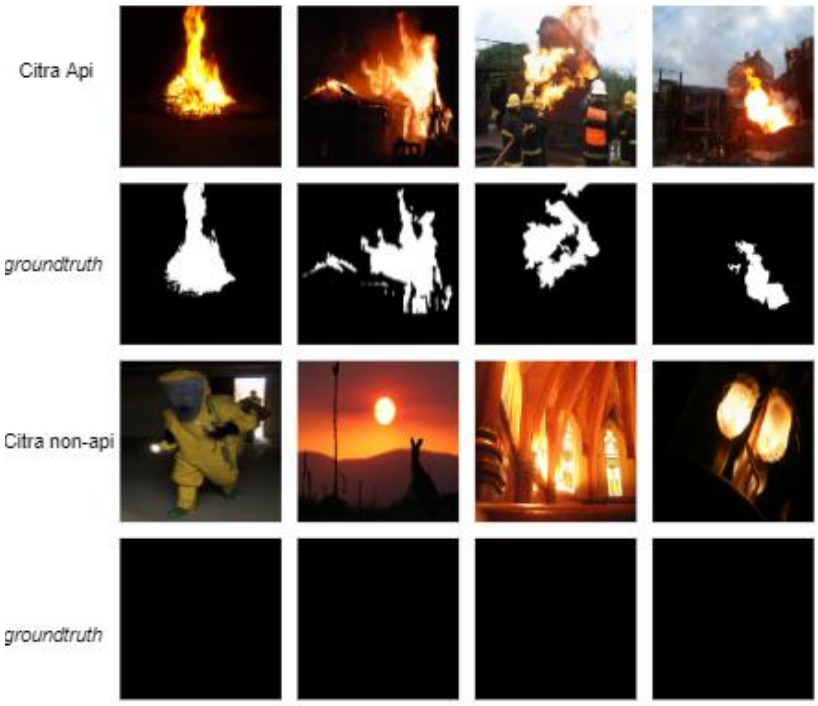
Namun untuk penelitian ini setidaknya dibutuhkan 3 dataset berbeda untuk kebutuhan *training set*, *test set* dan *training set* tambahan untuk diuji dalam pembahasan. Maka demikian untuk test set dan *training set* tambahan akan dterapkan pembuatan *ground truth* secara manual yang dijelaskan pada bagian 4.3.

Selain itu ketiga dataset tersebut tidak mempunyai dimensi citra yang seragam namun arsitektur yang diusulkan dan yang akan dibandingkan membutuhkan ukuran dimensi yang tetap pada proses pelatihannya. Untuk mengatasi masalah tersebut akan dilakukan prapemrosesan terhadap dimensi dataset.

Pra-pemrosesan yang digunakan adalah penyesuaian ukuran gambar. Umumnya pra-pemrosesan juga dapat dilakukan dengan *cropping* (pemotongan citra sampai dimensi yang ditentukan) namun hal ini akan menghilangkan informasi local maupun global pada sebuah citra. Maka karena itu *resize*(perubahan ukuran) dengan cara mengecilkan atau memperlebar dimensi citra dipilih sebagai metode pra-pemrosesan.

4.1.1 *BowFire*

Dataset yang dipilih sebagai *training set* pada penelitian ini adalah BowFire (*Best of both Worlds Fire detection*) dataset. Dataset ini terdiri dari 117 citra dengan objek api dan 109 citra dengan objek non api. Dataset ini dilengkapi dengan *ground truth* baik untuk citra api maupun yang bukan. Dataset ini dikembangkan oleh Chino et al.(2015) untuk penelitiannya mengenai pendeteksian api dengan mengintegrasikan analisis tekstur dan warna pixel pada citra.



Gambar 4.1 Dataset BowFire

4.1.2 *Fire Detection Image*

Dataset kedua yang digunakan berasal dari https://github.com/cair/Fire-Detection-Image-Dataset. Dataset ini dibuat oleh *Centre for Artificial Intelligence Research (CAIR)* di Universitas Agder, Norwegia. Dataset ini dibuat untuk menrefleksikan keadaan dunia nyata dengan citra api yang bervariatif pencahayaan, intesitas, ukurannya dan lingkungannya.

Gambar 4.2 Sampel dataset *Fire Detection Image*

Dataset ini tidak disertai dengan data citra *ground truth* sehingga perlu dilakukan pemrosesan lebih lanjut pada citra untuk mendapatkan *ground truth*. Jumlah total dataset ini adalah 651 citra yang terdiri dari 110 citra api dan 541 citra bukan api. Untuk penelitian ini dipilih 97 citra api dan 82 citra non api untuk dijadikan test set.

Pemilihan citra pada pada dataset ini didasarkan pada tingkat kesulitan pada proses pembuatan *ground truth* untuk citra api. 13 citra api yang tidak disertakan sebagai dataset mengandung objek yang tidak dapat dipisahkan antara api dan objek sekitarnya saat proses pembuatan *ground truth*. Hal ini ditakutkan mengurangi akurasi model saat proses pelatihan karena *bias* yang disebabkan oleh tidak akuratnya *ground truth*.

Pada citra non api, pemilihan dilakukan berdasarkan relevansi objek benda mirip api, Hanya 97 Citra non api dipilih dari 541 karena relevansinya dengan objek mirip api lebih baik daripada sisanya. Relevansi pada konteks ini adalah seberapa tinggi kemungkinan model akan mengenali citra sebagai *false positive* ataupun *false negative*.

Sebagian besar dataset untuk citra non api (disebut sebagai *Normal Images)* pada datasetnya tidak mencerminkan keadaan yang akan memacu *false positive*, diantaranya citra dengan objek danau biru, kantor dengan background putih dan pemandangan pulau. Untuk pemilihan yang dirasa sesuai dengan relevansinya adalah citra dengan sifat mirip api pada aspek warna dan bentuk seperti lampu dengan spektrum cahaya oranye, ruangan dengan pencahayaan dari dengan kombinasi oranye dan merah.

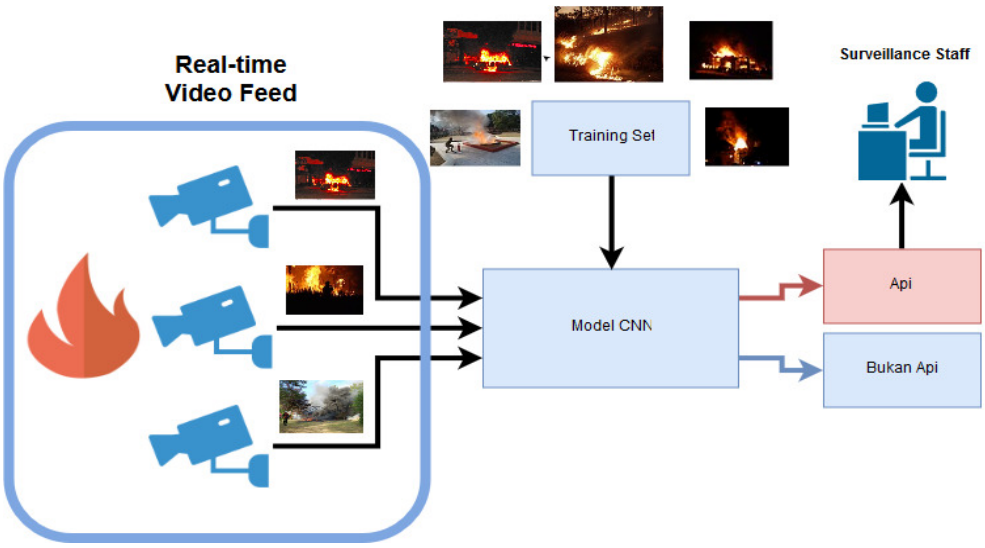
4.1.3 *Fire Smoke*

Dataset terakhir yang digunakan untuk menambah *training set* berasal dari *https://github.com/DeepQuestAI/Fire-Smoke-Dataset*. Sama seperti dataset kedua dataset ini tidak memiliki *groundtruth* sehingga perlu dilakukan pemrosesan lebih lanjut untuk mendapatkannya. Dataset ini terdiri dari 3000 citra dengan 3 kategori, yaitu api, asap dan normal. Setiap kategori dari citra mempunyai 900 citra untuk train set dan 100 citra untuk test set. Dari dataset ini dipilih 204 citra yang terdiri dari 104 citra dengan api dan 100 citra bukan api.

Gambar 4.3 Sampel Dataset *Fire Smoke*

Sama seperti dengan dataset *Fire Detection Image*, pemilihan citra api untuk train set didasari atas tingkat kesulitan pembuatan *ground truth*. Selain hal tersebut didapatkan banyak duplikat dari 900 citra api sehinga duplikat tidak disertakan. Dari kedua pertimbangan tersebut didapatkan 104 citra api.

4.2 Pengembangan arsitektur CNN



Gambar 4.4. Arsitektur sistem yang diusulkan

Arsitektur yang akan dibuat akan mendapatkan input real-time dari CCTV berbentuk video yang kemudian akan dianalisa per *frame* pada model CNN yang

sudah di *train* dari dataset yang sudah ada. Kemudian Model CNN akan mendeteksi apakah citra yang didapatkan mempunyai indikasi adanya api atau tidak.

Penggunaan Convolutional Neural Network pada deteksi api lebih banyak dilakukan dengan menggunakan arsitektur berdasarkan GoogLeNet dan AlexNet. Hasil penelitian Gonzales *et al.* menunjukkan bahwa implementasi *fully convolutional network* pada *AlexNet* yang dimodifikasi menunjukkan hasil yang signifikan. Dengan demikian penelitian ini akan merancang arsitektur *VGGNet* yang sudah dimodifikasi menjadi *fully convolutional network* yang dirancang oleh Long *et al.* dengan nama FCN-8s. Keluaran dari *fully convolutional network* akan berupa citra yang mempunyai segmentasi dalam tingkat *pixel* untuk api dan bukan api.

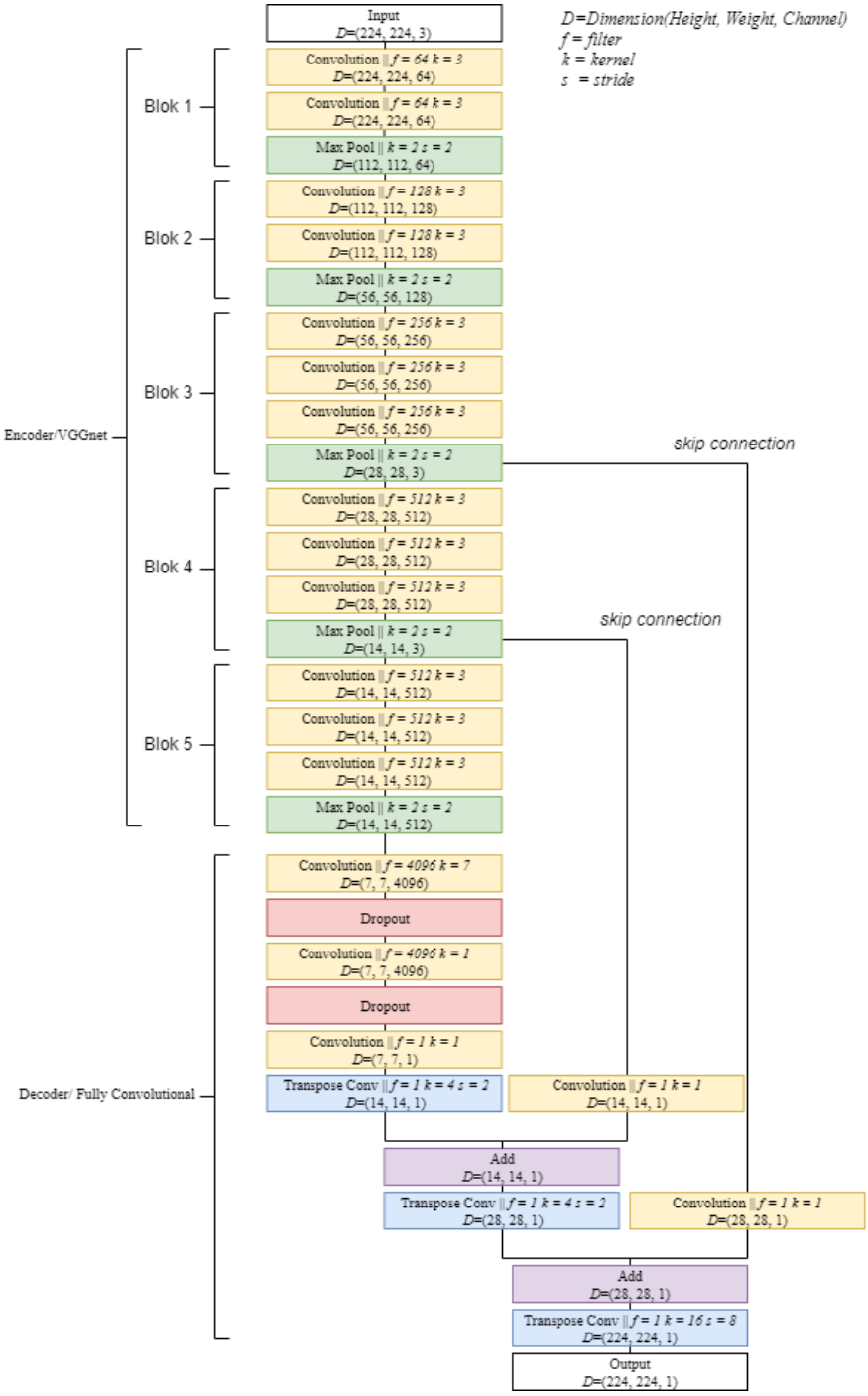
4.2.1 *FCN-8s*

Fully Convolutional Network 8 upsampled Prediction (FCN-8s) adalah arsitektur untuk menyelesaikan masalah segmentasi semantik yang diusulkan Long et al. (2015).Pada penelitiannya Long et al. mengubah arsitektur AlexNet, VGGNet dan GoogLeNet dengan membuang blok klasifikasi yang berupa fully connected layer pada setiap arsitektur dan mengubahnya menjadi *convolutional layer* dan *deconvolutional layer* pada layer terakhir untuk mendapatkan *label map*.

Dari 3 arsitektur *state-of-the-art* yang diubah, VGGnet mempunyai nilai rerata IU paling tinggi. Keluaran FCN-VGGNet ini masih berupa citra dengan segmentasi yang kasar. Hal ini terjadi karena masih ada detail informasi yang hilang dari proses konvolusi pada blok-blok sebelumnya sehingga digunakan *skip connection* dari pooling layer pada encoder untuk digabungkan dengan hasil konvolusi transposisi pada decoder.

Long *et al.* menjabarkan 3 bentuk FCN-VGGNet yang sudah dimodifikasi untuk kebutuhan segmentasi semantic: FCN-32s, FCN-16s, dan FCN-8s. FCN-32s adalah model VGGNet dengan menggantikan 2 *fully-connected layer* dengan *convolutional layer* serta menggantikan layer terakhri dengan *transpose convolutional layer* (disebut *deconvolutional layer* pada publikasinya). Berikutnya FCN-16s adalah model FCN-32s yang ditambah dengan skip connection dari blok encoder ke 4 untuk menambah detail dan kompleksitas.

Model terakhir yang mempunyai metrik terbaik dari publikasi Long *et al.* adalah FCN-8s. FCN-8s adalah FCN-16s dengan skip connection tambahan yang berasal dari blok 3 *encoder*. Gambar 4.5 merupakan spesifikasi secara umum arsitektur dari FCN-8s yang akan dipakai pada penelitian ini.



Gambar 4.5 Arsitektur FCN-8s

Encoder

Encoder pada FCN-8s adalah arsitektur VGGNet tanpa *fully-connected layer*. *Encoder* terdiri dari susunan 5 blok kombinasi *convolutional layer* yang dipisahkan dengan pooling layer. Terdapat total 13 *convolutional layer* dan 5 *pooling layer* pada encoder. Hyperparameter pada setiap layer sama kecuali jumlah filter yang berjumlah 64 dan kelipatannya sampai 512 pada blok terakhir.

Tabel 4.1 Parameter *Encoder*

Blok	Nama	*Filter*	Ukuran *Kernel*	*Stride*	*Padding*	Fungsi Aktivasi
Blok 1	*Conv 1.1*	64	3x3	1	*Same*	*ReLU*
	Conv 1.2					
	Block1_maxpool		2x2	2		
Blok 2	*Conv 2.1*	128	3x3	1	*Same*	*ReLU*
	Conv 2.2					
	Block2_maxpool		2x2	2		
Blok 3	*Conv 3.1*	256	3x3	1	*Same*	*ReLU*
	Conv 3.2					
	Conv 3.3					
	Block3_maxpool		2x2	2		
Blok 4	*Conv 4.1*	512	3x3	1	*Same*	*ReLU*
	Conv 4.2					
	Conv 4.3					
	Block4_maxpool		2x2	2		
Blok 5	*Conv 5.1*	512	3x3	1	*Same*	*ReLU*
	Conv 5.2					
	Conv 5.3					
	Block5_maxpool		2x2	2		

Berikut rincian dari layer pada *encoder*:

1. *Convolutional Layer*

Setiap blok pada encoder mempunyai *convolutional layer* berlipat dengan parameter yang sama dan jumlah filter yang berbeda. Filter dengan ukuran 3x3 bertujuan mengurangi jumlah parameter dan mendapatkan *effective receptive field* yang sama dengan 1 convolutional layer dengan ukuran kernel 7x7 (Simonyan dan Zisserman,2014). Setiap convolutional layer juga diikuti dengan fungsi aktivasi

ReLU, penggunaan fungsi aktivasi ReLU menambah nonlinearitas pada model yang berarti model dapat mengenali fitur yang lebih kompleks.

2. *Pooling Layer*

Pooling layer menggunakan fungsi *MaxPooling* dengan stride dan ukuran kernel yang sama pada setiap blok. Operasi *MaxPooling* dengan ukuran 2x2 dan stride 2 akan membuat dimensi citra menjadi setengah dari dimensi awal. Citra awalnya berdimensi 224x224 dan berakhir dengan ukuran 7x7 pada blok terakhir *Encoder*.

Decoder

Secara intuitif tujuan utama dari decoder adalah mengubah feature map yang didapatkan menjadi citra kelabu utuh dengan pixel-pixel yang sudah tersegmentasi antara api ataupun bukan api. Untuk mencapai hal tersebut dibutuhkan *upsampling* dari feature map. Proses *upsample* ini dilakukan dengan menggunakan *Transpose Convolutional layer* (lebih popular disebut dengan *deconvolutional layer*). Bagian *decoder* juga melakukan teknik *skip connection*, yaitu teknik menggabungkan layer tertentu dengan melewati layer setelahnya dengan tujuan mengatasi detail yang hilang pada proses encoding.

Tabel 4.2 Parameter Decoder

Nama	*Filter*	Ukuran *Kernel*	Stride	*Padding*	Fungsi Aktivasi	*rate*
Conv 6	4096	7	1	*Same*	*ReLu*	
Dropout						0.5
Conv 7	4096	1	1	*Same*	*ReLu*	
Dropout						0.5
Conv 8	1	1	1	*Same*	*ReLu*	
Transpose_conv1	1	4	2	*Same*	*ReLu*	
Block3_fullyconv	1	1	1	*Same*	-	
Add						
Transpose_conv2	1	4	2	*Same*	*ReLu*	
Block4_fullyconv	1	1	1	*Same*	-	
Add						
Transpose_conv3	1	16	8	*Same*	*Sigmoid*	

Berikut rincian *decoder*:

1. *Convolutional Layer*

Terdapat 5 convolutional layer pada decoder, tiga layer pertama (*conv6, conv7* dan *conv8*) adalah *fully-connected layer* yang diubah menjadi *convolutional layer*. Kebutuhan untuk mengubah fully connected layer menjadi convolutional layer adalah mengakomodasi ukuran citra yang dynamis, hal ini tidak bisa dilakukan oleh *fully-connected layer* karena layer ini membutuhakn ukuran masukan yang tetap. Dua layer berikutnya adalah bentuk *skip connection* yang berasal dari *pooling layer* pada encoder, yaitu *block3_maxpool* dan *block4_maxpool*. *Block3_fullyconv* dan *Block4_fullyconv* digunakan untuk mendapatkan kompleksitas dan detail yang hilang dari proses encoding dengan menggabungkan hasil layer tersebut *element-wise* pada pada *transpose convolutional layer*.

2. *Dropout Layer*

Dua buah dropout layer di pasang di antara setelah *conv6* dan *conv7* dengan *rate* 0.5. *Rate* sebesar 0.5 berarti 50% dari seluruh jumlah nodes pada masing-masing *conv6* dan *conv7* akan dibuang. Pembuangan ini dimaksudkan untuk mengurangi *overfitting* yang berpotensi terjadi saat proses pelatihan.

3. *Add Layer*

Add Layer digunakan untk menggabungkan hasil transpose convolutional layer dan skip connection dari encoder. *Add layer* berfungsi melakukan operasi *element-wise sum* pada kedua buah feature map yang dihasilkan. Syarat untuk kedua buah layer agar bisa dikenakan fungsi *add* adalah keduanya harus mempunyai dimensi keluaran yang sama sehingga keluaran dari add layer tidak merubah dimensi.

4. *Transpose Convolutional Layer*

Transpose convolutional layer pada dasarnya melakukan fungsi *upsampling* pada masukan feature map. Digunakan tiga buah *transpose convolutional layer* pada arsitektur. *Transpose_conv1* mengubah dimensi feature map dari conv_8 menjadi 14x14. Namun hasil dari *Transpose_conv1* masih sangat kasar sehingga digabungkan lah *Transpose_conv1* dengan *block4_fullyconv* dengan add layer pertama yang kemudian hasil dari add layer tersebut dilanjutkan menuju *transpose_conv2* sehingga dimensinya berubah menjadi 28x28. *Transpose_conv2*

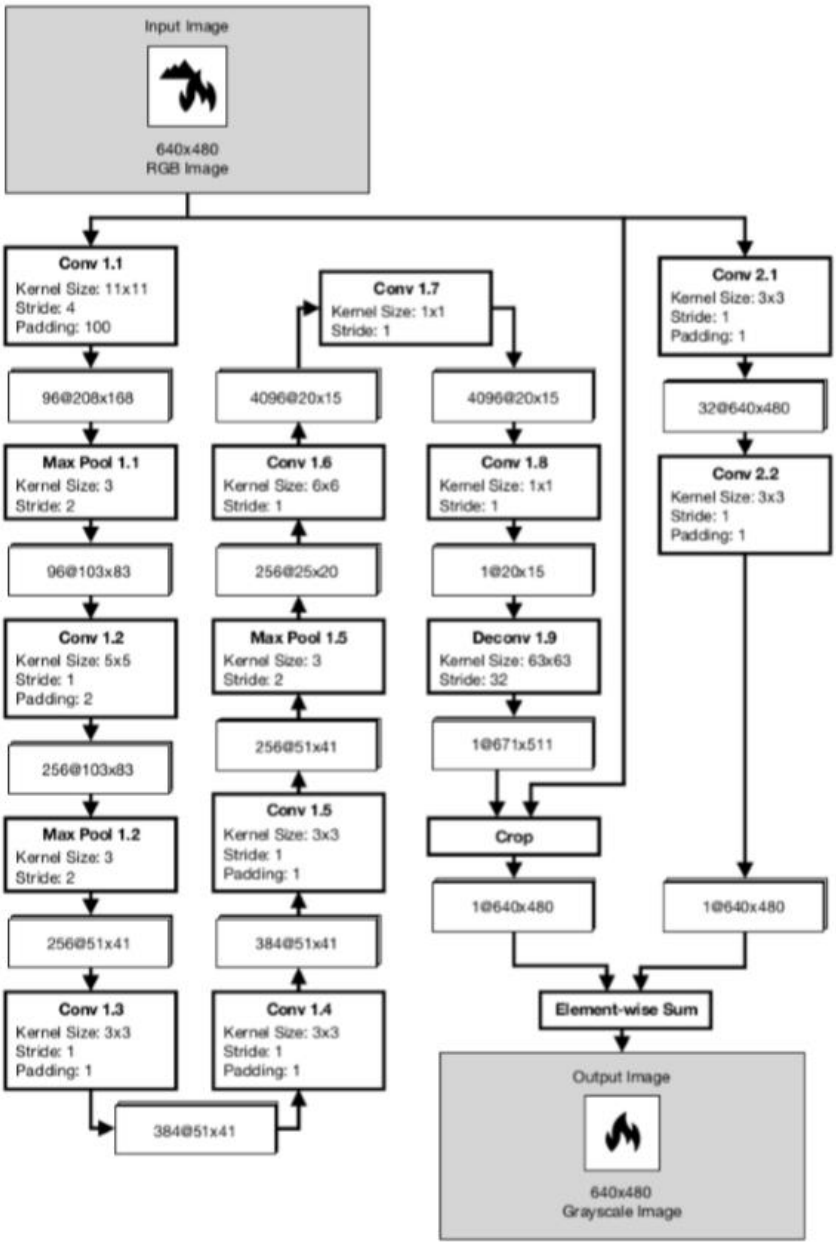
kemudian digabungkan lagi dengan *block3_fullyconv* untuk menambah kompleksitas dan detail yang hilang dari proses *upsampling*. *Feature map* terakhir yang dihasilkan kemudian dilanjutkan menuju layer terakhir yaitu *transposed_conv3*. Hasil akhir dari layer ini adalah larik 2D dengan dimensi 224x224. Berbeda dengan *layer-layer* sebelumnya yang menggunakan *ReLU*, fungsi aktivasi pada layer *transpose_conv3* menggunakan *sigmoid*.

4.2.2 *Simple Feature Extraction with FCN AlexNet, Single Deconvolution (SFEwAN-SD)*

Penelitian ini akan membandingkan hasil penelitian usulan dengan *Simple Feature Extraction with AlexNet, Single Deconvolution* yang diusulkan oleh Gonzaleset al.(2017). SFEwAN-SD merupakan integrasi dari dua arsitektur. Pertama adalah AlexNet yang dimodifikasi dengan tujuan mendapatkan bentuk dan tekstur api. Pada bagian akhir arsitektur ini dilakukan operasi *deconvolution* untuk mengubah feature map menjadi citra. Arsitektur pertama pada gambar 4.6 ditunjukkan dengan layer berawalan 1. Arsitektur kedua adalah runtutan 2 *convolutional layer* dari citra masukan untuk mendapatkan fitur warna dan tekstur.arsitektur kedua ditandai dengan convolutional layer berawalan 2. Setelah itu Kedua arsitektur ini kemudian digabungkan pada keluaran.

Penggabungan ini menurut Gonzales et al. bertujuan menghindari kekurangan dari masing-masing arsitektur. Arsitektur pertama mampu mendeteksi api namun kehilangan resolusi asli karena proses dekonvolusi. Arsitektur kedua dapat mendeteksi api namun mempunyai tingkat *false positive* yang tinggi.

Baik FCN-8s dan SFEwAN-SD menggunakan *skip connection* sebagai cara menambah detail yang hilang dari arsitektur utama. SFEwAN-SD pada publikasiya menyebut *skip connection* ini sebagai arsitektur kedua. Perbedaan mendasar penggunaan *skip connection* pada FCN-8s dan SFEwAN-SD adalah posisi pengambilan *skip connection.* FCN-8s menggunaka 2 skip connection yang berasal dari *max pooling layer* sedangkan SFEwAN-SD menggunakan skip connection langsung dari citra masukan. Penelitian ini juga akan mencoba melihat pengaruh skip connection pada akurasi model dengan menambah 1 skip connection pada SFEwAN-SD dari salah satu dari 3 *pooling layer*.

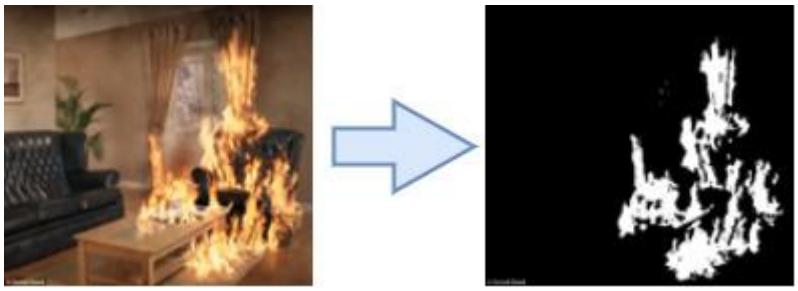


Gambar 4.6 SFEwan-SD

4.3 Pembuatan Ground Truth

Pembuatan *ground truth* pada citra dilakukan dengan fitur masking manual menggunakan perangkat lunak GIMP dengan teknik thresholding pada value warna merah dengan jangkauan warna merah bervariasi antara 200-255 sesuai dengan kebutuhan pada gambar. Jangkauan warna merah pada citra bervariasi karena adanya perbedaan resolusi dan intensitas api. Proses *thresholding* mengisolasi 1 kanal warna yang diinginkan dan mengubah citra menjadi format gray scale dimana

hanya ada 2 nilai (0,255). Jangkauan warna merah pada citra bervariasi karena adanya perbedaan resolusi dan intensitas api.



Gambar 4.7 Perubahan Citra Warna menjadi Ground Truth

4.4 Perancangan Pelatihan

Pelatihan rencananya akan dilakukan pada platform *Google Colab* karena ketersediaaan GPU dan RAM dengan spesifikasi yang lebih tinggi daripada GPU dan RAM laptop pada umumnya. Spesifikasi yang lebih tinggi memungkinkan model untuk memuat sampel lebih banyak untuk setiap batch dan memangkas waktu pelatihan secara signifikan. Berikut parameter tetap untuk pelatihan:

Tabel 4.3 Hyperparameter pelatihan

Optimizer	*Adam* dengan *learning rate* = 10^{-3}
Loss	*Binary Cross-entropy*
Metric	*Binary Accuracy*
Epoch	200
Batch Size	32

4.5 Perancangan Evaluasi Model

Metriks evaluasi untuk penelitian ini akan menggunakan *Confusion matrix*. *Confusion matrix* digunakan karena penelitian ini adalah klasifikasi biner (Deteksi citra dengan api dan tanpa api) serta dapat menunjukan performa klasifikasi model dari nilai akurasi, *F1-score* serta *mean Intersection-over-Union*(*mIOU*). *Confusion matrix* mempunyai empat jenis kelas berdasarkan hasil prediksi dan observasi. Empat hal tersebut yaitu :

1. *True Positive (TP)*

Kelas dengan hasil prediksi ada api dan observasi benar ada api

2. *True Negative (TN)*

Kelas dengan hasil prediksi tidak ada api dan observasi tidak ada api

3. *False Positive (FP)*

Kelas dengan hasil prediksi ada api namun observasi tidak ada api

4. *False Negative (FN)*

Kelas dengan hasil prediksi tidak ada api namun observasi ada api

Akurasi model ditentukan dengan formula berikut :

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

Skor F1 model akan ditentukan dengan formula berikut:

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

Di mana $Recall = \frac{TP}{TP+FN}$ dan $Precision = \frac{TP}{TP+FP}$

Mean Intersection-over-Union akan ditentukan dengan formula berikut:

$$mean\ Intersection - over - Union\ = \frac{TP}{(TP + FP + FN)}$$

BAB V
IMPLEMENTASI

5.1 Perangkat Penelitian

5.1.1 *Perangkat Lunak*

Metode pengembangan akan dilakukan melalui Google Colaboratory. Google Colaboratory adalah layanan pengembangan *machine learning* berbasis *cloud* untuk kebutuhan edukasi dan penelitian yang disediakan secara gratis oleh Google. Penelitian akan dilakukan dengan detail sebagai berikut :

Bahasa Pemrograman : *Python*

Environment : *Jupyter Notebook*

Library : *Keras, Numpy, Tensorflow*

Framework : *Tensorflow*

5.1.2 *Perangkat Keras*

Spesifikasi perangkat keras yang disediakan Google Colaboratory mempunyai spesifikasi sebagai berikut :

CPU : Intel(R) Xeon(R) CPU @ 2.20GHz

GPU : NVIDIA Tesla K80

RAM :13 GB

5.2 Pra-pemrosesan Dataset

5.2.1 *Perubahan Citra menjadi Larik*

Format citra yang menjadi masukan pada *Tensorflow* berupa larik numpy dengan ukuran 4D dengan bentuk (*batch size*, Panjang, lebar, kanal) di mana *batch size* mengacu pada jumlah masukan yang akan dimasukkan pada model. Perubahan citra menjadi larik menggunakan library *Tensorflow* load_img dan img_as_array serta library numpy.save.

```
import os
import numpy as np
from Tensorflow.python.keras.preprocessing.image import load_
img, img_to_array
img_path='C:/BowFire/dataset/img_new/'
mask_path='C:/BowFire/dataset/gt_new/'
```

Gambar 5.1 Implementasi perubahan citra menjadi larik (1)

```
IMAGE_WIDTH=224
IMAGE_HEIGHT=224
IMG_CHANNELS=3
MASK_CHANNEL=1

img_list = os.listdir(img_path)
img_file=np.array(img_list)
num_files=len(img_list)

mask_list = os.listdir(mask_path)
mask_file=np.array(mask_list)
mask_files=len(mask_list)


X = np.zeros((num_files,IMAGE_HEIGHT, IMAGE_WIDTH, IMG_CHANNE
LS), dtype=np.uint8)
Y = np.zeros((num_files,IMAGE_HEIGHT, IMAGE_WIDTH, MASK_CHANN
EL), dtype=np.uint8)

for n in range(num_files):
    filenumber=str(n).zfill(3)
    img_dir=img_path+img_file[n]
    img = load_img(img_dir,target_size=(IMAGE_HEIGHT,IMAGE_HE
IGHT))
    train_img = img_to_array(img)
    X[n] = train_img
    mask_dir=mask_path+mask_file[n]
    mask = load_img(mask_dir,target_size=(IMAGE_HEIGHT,IMAGE_
HEIGHT),color_mode='grayscale')
    train_mask = img_to_array(mask)
    Y[n] = train_mask
np.save('image.npy',X)
np.save('label.npy',Y)
```

Gambar 5.2 Implementasi perubahan citra menjadi larik (2)

5.2.2 *Normalisasi Dataset*

```
X=X/255.
Y=Y.astype(np.bool)
```

Gambar 5.3 Implementasi Normalisasi dataset

Normalisasi dataset diperlukan untuk mengurangi beban komputasi saat proses pelatihan. Tanpa normalisasi proses pelatihan dapat berjalan namun komputasi yang diperlukan semakin tinggi. Beban komputasi juga akan berefek pada jumlah *batch size* pada proses pelatihan. Semakin besar nilai suatu larik maka semakin sedikit jumlah sampel yang bisa dimasukkan karena keterbasan memori.

5.3 Implementasi *Deep Learning*

5.3.1 Arsitektur *FCN-8s*

Implementasi FCN-8s asli menggunakan framework deep learning Caffee namun pada penelitian ini digunakan framework *Tensorflow* dengan library *Keras* karena proses implementasi lebih mudah dibandingkan Caffee. Implementasi FCN-8s dioptimalkan untuk dataset *PASCAL Visual Object Challenge* (VOC) yang mempunyai 20 kelas objek segmentasi. Implementasi untuk penelitian ini diubah untuk menyesuaikan tujuan penelitian dan dataset yang digunakan.

Bila pada PASCAL-VOC tujuan dari segmentasi semantik adalah untuk segmentasi multilabel dalam gambar, penelitian ini segmentasi semantik yang diinginkan berupa 2 kelas atau klasifikasi biner di mana hanya ada kelas api dan non-api dengan *ground truth* berupa citra kelabu. Dengan pertimbangan tersebut, jika pada implementasi layer terakhir digunakan fungsi aktivasi *Softmax* pada penelitian asli maka digunakan fungsi aktifasi *sigmoid* pada penelitian karena lebih optimal digunakan untuk klasifikasi biner.

Long *et al.* menyebutkan pada publikasinya bahwa operasi yang digunakan untuk *upsampling* disebut sebagai *backward convolution* atau lebih umum disebut *deconvolution*. *Library Keras* tidak mempunyai *layer* dengan nama *deconvolution* atau *backward convolution*, namun *layer dengan* definisi dan fungsi yang sama disebut sebagai *Transpose Convolutional layer*. *Library Keras* yang digunakan pada implementasi ini adalah Conv2DTranspose.

```
import Tensorflow as tf
from Keras import applications
from Keras.models import *
import skimage.io as io
import skimage.transform as trans
from Keras.layers import *
from Keras.optimizers import *
from Keras.callbacks import ModelCheckpoint
from Keras import backend as Keras

image_size=[224,224]
ch_in=3
num_class=1
```

Gambar 5.4 FCN-8s (1)

```
inputs=Input(shape=(*image_size, ch_in), name='input_layer')
#block1
c11=Conv2D(64, (3, 3), activation='relu', padding='same', name='
block1_conv11')(inputs)
c12=Conv2D(64, (3, 3), activation='relu', padding='same', name='
block1_conv12')(c11)
mp1=MaxPooling2D((2, 2), strides=(2, 2), name='block1_maxpool')(
c12)
#block2
c21=Conv2D(128, (3, 3), activation='relu', padding='same', name=
'block2_conv21')(mp1)
c22=Conv2D(128, (3, 3), activation='relu', padding='same', name=
'block2_conv22')(c21)
mp2=MaxPooling2D((2, 2), strides=(2, 2), name='block2_maxpool')(
c22)
#block3
c31=Conv2D(256, (3, 3), activation='relu', padding='same', name=
'block3_conv31')(mp2)
c32=Conv2D(256, (3, 3), activation='relu', padding='same', name=
'block3_conv32')(c31)
c33=Conv2D(256, (3, 3), activation='relu', padding='same', name=
'block3_conv33')(c32)
mp3=MaxPooling2D((2, 2), strides=(2, 2), name='block3_maxpool')(
c33)
fc3 = Conv2D(num_class,(1,1), activation=None,padding='same', na
me='block3_fullyconv')(mp3)
#block4
c41=Conv2D(512, (3, 3), activation='relu', padding='same', name=
'block4_conv41')(mp3)
c42=Conv2D(512, (3, 3), activation='relu', padding='same', name=
'block4_conv42')(c41)
c43=Conv2D(512, (3, 3), activation='relu', padding='same', name=
'block4_conv43')(c42)
mp4=MaxPooling2D((2, 2), strides=(2, 2), name='block4_maxpool')(
c43)
fc4 = Conv2D(num_class,(1,1), activation=None,padding='same', na
me='block4_fullyconv')(mp4)
#block5
c51=Conv2D(512, (3, 3), activation='relu', padding='same', name=
'block5_conv51')(mp4)
c52=Conv2D(512, (3, 3), activation='relu', padding='same', name=
'block5_conv52')(c51)
c53=Conv2D(512, (3, 3), activation='relu', padding='same', name=
'block5_conv53')(c52)
mp5=MaxPooling2D((2, 2), strides=(2, 2), name='block5_maxpool')(
c53)
```

Gambar 5.5 FCN-8s (2)

```
#(Fully Convolutional)
c6=Conv2D(4096, (7, 7), activation='relu', padding='same', name=
'conv6')(mp5)
d6=Dropout(0.5)(c6)
c7=Conv2D(4096, (1, 1), activation='relu', padding='same', name=
'conv7')(d6)
d7=Dropout(0.5)(c7)
c8=Conv2D(num_class, (1, 1),activation=None, padding='same', nam
e='conv8')(d7)
tc1=Conv2DTranspose(num_class, (4, 4), strides=(2,2), padding='s
ame', name='transpose_conv1')(c8)
add1=add([fc4,tc1],name='Sum_1')
tc2=Conv2DTranspose(num_class, (4, 4), strides=(2,2), padding='s
ame', name='transpose_conv2')(add1)
add2=add([fc3,tc2],name='Sum_2')
tc3=Conv2DTranspose(num_class, (16,16), strides=(8,8), padding='
same', name='transpose_conv3')(add2)
output=Activation('sigmoid',name='output_layer')(tc3)
```

Gambar 5.6 FCN-8s (3)

Gambar 5.6 menunjukkan bagian *Decoder* FCN-8s yang berfungsi mengubah feature map yang didapatkan dari *Encoder* menjadi citra tersegmentasi. Dibagian ini digunakan layer Conv2D, Dropout, dan Conv2DTranspose. Tahap Transpose_conv3 disertai fungsi aktivasi sigmoid sebagai output layer.

5.3.2 Arsitektur *SFEwAN-SD*

Arsitektur Simple Feature Extraction with FCN AlexNet, Single Deconvolution (SFEwAN-SD) diimplementasikan hanya dengan referensi model yang ditampilkan pada penelitian Gonzales *et al.*(2017). Karena tidak ada publikasi kode dan dataset yang digunakan pada penelitiannya. Parameter yang didapatkan dari penelitian berupa bentuk input, fungsi aktivasi, *padding*, *kernel size*, *stride* dan jenis *layer*.

```
from Keras.models import *
from Keras.layers import *
from Keras.optimizers import *
from Keras.callbacks import ModelCheckpoint
from Keras import backend as Keras
image_size=[640,480]
ch_in=3
num_class=1
x=Input(shape=(*image_size, ch_in), name='input_layer')
```

Gambar 5.7 SFEwAN-SD (1)

```
#First Part
#block 1
x1=ZeroPadding2D(100,name='Padding_1')(x)
c11=Conv2D(96, (11, 11),strides=4, activation='relu', name='C
onv_1.1')(x1)
#C11 kernel size 11x11 stride 4 padding 100 96 filter
mp11 = MaxPooling2D((3, 3), strides=(2, 2), name='MaxPool_1.1
')(c11)
#maxpool 11 kernel size 3 stride 2
#block 2
mp11_1 = ZeroPadding2D(2,name='Padding_2')(mp11)
c12 = Conv2D(256,(5, 5), strides=1, activation='relu', name='
Conv_1.2')(mp11_1)
#C12 kernel size 5x5 stride 1 padding 2 256 filters
mp12 = MaxPooling2D((3, 3), strides=(2, 2), name='MaxPool_1.2
')(c12)
#maxpool12 2 kernel size 3 stride 2
c13 = Conv2D(384,(3, 3), strides=1, padding = 'same', activat
ion='relu', name='Conv_1.3')(mp12)
#C13 kernel size 3x3 stride 1 padding 1 384 filters
c14 = Conv2D(384,(3, 3), strides=1, padding = 'same', activat
ion='relu', name='Conv_1.4')(c13)
#C14 kernel size 3x3 stride 1 padding 1 384 filters
c15 = Conv2D(256,(3, 3), strides=1, padding = 'same', activat
ion='relu', name='Conv_1.5')(c14)
#C15 kernel size 3x3 stride 1 padding 1 256 filters
mp15 = MaxPooling2D((3, 3), strides=(2, 2), padding = 'valid'
, name='MaxPool_1.5')(c15)
#Maxpool 15  kernel size 3 stride 2
c16 = Conv2D(4096,(6, 6), strides=1, padding = 'valid', activ
ation='relu', name='Conv_1.6')(mp15)
#C16 kernel size 6x6 stride 1 4096 filters
c17 = Conv2D(4096,(1, 1), strides=1, padding = 'valid', activ
ation='relu', name='Conv_1.7')(c16)
#C7 kernel size 1x1 stride 1 4096 filters
c18 = Conv2D(1,(1, 1), strides=1, padding = 'valid', name='Co
nv_1.8')(c17)
#C8 kernel size 1x1 stride 1 1 filters
dc19 = Conv2DTranspose(1, (63, 63), strides=32, padding = 'va
lid', name='Deconv_1.9')(c18)
#DC9 kernel size 63x63 stride 32 1 filter
cr1 = Cropping2D([(21,10),(21,10)], name='Crop_1')(dc19)
#CR 640x480
```

Gambar 5.8 SFEwAN-SD (2)

```
#Second Part
c21 = Conv2D(32,(3, 3), strides=1, padding = 'same', activati
on='relu', name='Conv_2.1')(x)
#C21 kernel 3x3 stride 1 padding 1
c22 = Conv2D(1,(3, 3), strides=1, padding = 'same', activatio
n='relu', name='Conv_2.2')(c21)
#C22 kernel 3x3 stride 1 padding 1
output= add([cr1,c22],name='Output')
gonzales_model=Model(inputs=x,outputs=output)
```

Gambar 5.9 SFEwAN-SD (3)

Gambar 5.10 adalah modifikasi penambahan skip connection dari tiga layer max pooling berbeda pada SFEwAN-SD. Setiap bagian diimplementasikan secara independen untuk pengujian. Implementasi setiap bagian menggunakan tiga layer yang sama yaitu Conv2D sebagai convolutional layer, Conv2Dtranspose untuk *upscale* dan Cropping2D untuk memotong dimensi hasil upscaling menjadi 640x480. Hasil dari skip connection ini dipadukan dengan output asli.

```
#SKIP CONNECTION MAX POOL 1.5
c31=Conv2D(1,(1, 1), strides=1, padding = 'same', name='Conv_
3.1')(mp15)
dc31 = Conv2DTranspose(1, (50, 49), strides=25, padding = 'va
lid', name='Deconv_3.1')(c31)
cr2= Cropping2D([5,22], name='Crop_2')(dc31)
#SKIP CONNECTION MAX POOL 1.1
c31=Conv2D(1,(1, 1), strides=1, padding = 'same', name='Conv_
3.1')(mp11)
dc31 = Conv2DTranspose(1, (14, 14), strides=7, padding = 'val
id', name='Deconv_3.1')(c31)
cr2= Cropping2D([58,68], name='Crop_2')(dc31)
#SKIP CONNECTION MAX POOL 1.2
c31=Conv2D(1,(1, 1), strides=1, padding = 'same', name='Conv_
3.1')(mp12)
dc31 = Conv2DTranspose(1, (26, 26), strides=13, padding = 'va
lid', name='Deconv_3.1')(c31)
cr2= Cropping2D([18,33], name='Crop_2')(dc31)

add1= add([cr1,c22],name='Sum1')
output=add([add1,cr2],name='output')
gonzales_model=Model(inputs=x,outputs=output)
```

Gambar 5.10 SFEwAN-SD (4)

5.4 Implementasi Proses Pelatihan

Implementasi Proses pelatihan dibagi menjadi 3 tahap. Dimulai dengan *dataset splicing* diikuti *model compling* dan diakhiri dengan *model fitting*. Khusus untuk dataset splicing digunakan *library scikit-learn* karena splicing menggunakan *Keras* tidak mengaplikasikan *shuffle* pada dataset. Implementasi proses pelatihan ini menggunakan parameter yang sama baik untuk FCN-8s dan SFEwAN-SD.

5.4.1 *Dataset Splicing*

```
from sklearn.model_selection import train_test_split
X_train, X_val, Y_train, Y_val = train_test_split(X, Y, test_
size=0.2, random_state=0)
```

Gambar 5.11 *Dataset Splicing*

Dataset diubah menjadi training set (X_train, Y_train) dan validation set(X_val, Y_val) untuk validasi akurasi dari model. Rasio training dan validation set adalah 80 % persen untuk training dan 20 % untuk validation set.

5.4.2 *Model Compiling*

```
model.compile(optimizer=Adam(), loss='binary_crossentropy',
metrics=['binary_accuracy'])
```

Gambar 5.12 *Model Compiling*

Fungsi utama dari *model.compile* adalah menetapkan konfigurasi dari proses pelatihan, parameter yang dipakai adalah *Optimizer, Loss* dan *Metrics*. FCN-8s aslinya menggunakan *Stochastic Gradient Descent* (SGD) sebagai optimizer namun hasil eksperimen proses pengurangan *loss* sangat lambat tiap epoch. Eksperimen menggunakan fungsi optimisasi Adam menunjukkan hasil yang lebih signifikan.

Loss dan *metrics* yang dipakai pada FCN-8s awalnya menggunakan Crossentropy dan Accuracy. Namun tidak optimal untuk klasifikasi biner sehingga binary crossentropy dan binary accuracy karena output yang diharapkan berada berada di antara [0,1].

5.4.3 Model Fitting

```
checkpointer = ModelCheckpoint(filepath='/content/
weights.hdf5', verbose=1, save_best_only=True)
model.fit(X_train,Y_train,validation_data=[X_test,Y_test],
batch_size=32,epochs=200,callbacks=[checkpointer])
```

Gambar 5.13 Model Fitting

Fit adalah method untuk mengatur proses pelatihan dengan parameter yang ditentukan. *Batch_size* adalah parameter yang mengatur jumlah sample yang diproses setiap model melakukan pembaharuan parameter internal. Umumnya batch size berkisar antar 1 sampai dengan jumlah total training set itu sendiri. Pada penelitian ini batch size pada penelitian ini ditentukan sejumlah 32 sample.

Epoch adalah jumlah iterasi sebuah model setiap proses pelatihan. *Epoch* yang ditentukan adalah 200, berarti model melakukan proses pembaharuan parameter internal menggunakan training set sebanyak 200 kali pada masa pelatihan. *Callbacks* adalah fungsi yang digunakan untuk mencatat perubahan perubahan parameter internal pada model. Parameter *Callback* yang digunakan pada pelatihan adalah *ModelCheckpoint* yang berfungsi menyimpan *weight* model dengan param *save_best_only* untuk menyimpan *weight* dengan *loss* paling kecil.

5.5 Implementasi Hasil Prediksi

```
X_test=X_test/255.
pred_test = model.predict(X_test)
pred_test[pred_test >= 0.5] = True
pred_test[pred_test<0.5] = False
```

Gambar 5.14 Implementasi hasil prediksi

Untuk mendapatkan hasil prediksi test set harus dinormalisasi terlebih dahulu. Setelah itu hasil normalisasi diproses melalui method *predict*. *Pred_test* didapatkan sebagai hasil prediksi yang kemudian diubah nilainya dengan nilai ambang 0.5 untuk menyesuaikan dengan format *true* dan *false*.

5.6 Implementasi Pengubahan Larik Menjadi Citra

```
from keras.preprocessing.image import save_img

for i in range(num_files):
  filenumber=str(i+1).zfill(3)
  filename='test_pred/fire'+filenumber+'.png'
  img=pred_test[i]
  save_img(filename,img)
```

Gambar 5.15 Implementasi pengubahan larik menjadi citra

Implementasi perubahan larik menjadi citra utamanya dilakukan untuk evaluasi pada tingkat citra. *Library save_img* yang digunakan pada dokumentasinya sudah mencakup *library array_to_img* sehingga tidak diperlukan fungsi

array_to_img sebelumnya. Proses perubahan larik menjadi citra ini mengubah larik 2 dimensi menjadi citra kelabu.

5.7 Implementasi Evaluasi

Evaluasi dilakukan dengan melihat perbandingan hasil akurasi dan F1-score setiap model menggunakan test dataset. Implementasi evaluasi menggunakan library scikit-learn untuk mendapatkan nilai true positive, true negatives, false positives dan false negatives. Hasil prediksi dari model adalah nilai dengan rentang [0,1] sehingga untuk untuk mendapatkan nilai biner True dan False diperlukan batas ambang untuk menentukan mana nilai yang sejatinya masuk kategori api atau bukan api digunakan nilai sebesar 0.5.

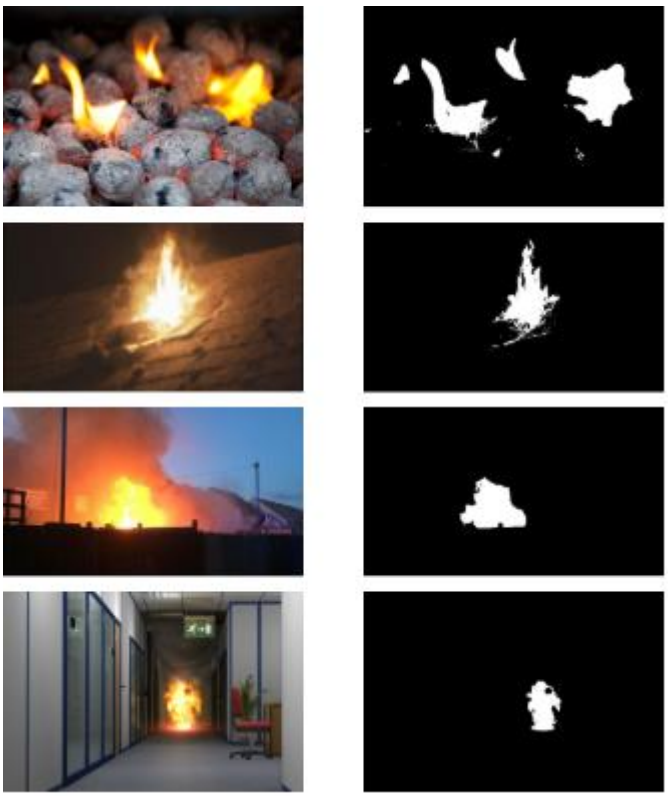
```
from sklearn.metrics import confusion_matrix
true_label = y_test.flatten()
true_pred = test_pred.flatten()
tn, fp, fn, tp=Confusion_matrix(true_label,true_pred).ravel()
print("True Positives  : ", tp)
print("True Negatives  : ", tn)
print("False Positives : ", fp)
print("False Negatives : ", fn)
accuracy =(tp+tn)/(tp+tn+fp+fn)
precision = tp/(tp+fp)
recall = tp/(tp+fn)
f1score =2*(recall*precision)/(recall+precision)
iou = tp/(tp+fp+fn)
print("Accuracy  : ",accuracy)
print("Precision : ",precision)
print("Recall    : ",recall)
print("F1-Score  : ",f1score)
print("mIOU      : ",iou)
```

Gambar 5.16 Implementasi evaluasi

BAB VI
HASIL DAN PEMBAHASAN

6.1 Hasil Pembuatan Ground Truth

Hasil pembuatan *ground truth* untuk seluruh dataset menggunakan perangkat lunak *GIMP* dibagi menjadi 2 bagian, citra api dan citra non api. Untuk citra api gambar diubah menjadi citra kelabu dengan mengisolasi kanal warna merah. Untuk citra dengan objek mirip atau berwarna api seperti pada gambar ketiga di gambar 6.1 diperlukan pembersihan citra untuk mendapatkan citra kelabu api yang tidak tercampur dengan objek bukan api seperti asap.



Gambar 6.1 Pembuatan *ground truth*

Pembuatan *ground truth* untuk citra non api dilakukan dengan menciptakan citra berwarna hitam dengan format *grayscale*. Setiap citra hitam disesuaikan ukurannya berdasarkan dimensi citra non api yang akan digunakan.

Gambar 6.2 Hasil pembuatan *ground truth* citra bukan api

6.2 Hasil Normalisasi Dataset

Dataset X (citra) dan Y(*groundtruth*) pada awalnya mempunyai rentang nilai [0,255] tanpa normalisasi dataset jalannya pelatihan akan akan berjalan lebih lamban karena karena proses komputasi yang relative sangat mahal. Normalisasi dilakukan dengan membagi nilai asli dari larik dengan nilai 255 untuk mengubah rentang nilai menjadi [0,1]. Untuk Dataset Y yang berisi mask atau groundtruth diubah menjadi Boolean untuk mengubah nilai 0 dan 255 menjadi *False* dan *True*.

```
[5] print(np.unique(X))
    print(np.unique(Y))
```

```
[  0   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17
  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35
  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53
  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71
  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89
  90  91  92  93  94  95  96  97  98  99 100 101 102 103 104 105 106 107
 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125
 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143
 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161
 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179
 180 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197
 198 199 200 201 202 203 204 205 206 207 208 209 210 211 212 213 214 215
 216 217 218 219 220 221 222 223 224 225 226 227 228 229 230 231 232 233
 234 235 236 237 238 239 240 241 242 243 244 245 246 247 248 249 250 251
 252 253 254 255]
[  0 255]
```

Gambar 6.3 Dataset sebelum normalisasi

```
[7] print(np.unique(X))
    print(np.unique(Y))
```

```
[0.         0.00392157 0.00784314 0.01176471 0.01568627 0.01960784
 0.02352941 0.02745098 0.03137255 0.03529412 0.03921569 0.04313725
 0.04705882 0.05098039 0.05490196 0.05882353 0.0627451  0.06666667
 0.07058824 0.0745098  0.07843137 0.08235294 0.08627451 0.09019608
 0.09411765 0.09803922 0.10196078 0.10588235 0.10980392 0.11372549
 0.11764706 0.12156863 0.1254902  0.12941176 0.13333333 0.1372549
 0.14117647 0.14509804 0.14901961 0.15294118 0.15686275 0.16078431
 0.16470588 0.16862745 0.17254902 0.17647059 0.18039216 0.18431373
 0.18823529 0.19215686 0.19607843 0.2        0.20392157 0.20784314
 0.21176471 0.21568627 0.21960784 0.22352941 0.22745098 0.23137255
 0.23529412 0.23921569 0.24313725 0.24705882 0.25098039 0.25490196
 0.25882353 0.2627451  0.26666667 0.27058824 0.2745098  0.27843137
 0.28235294 0.28627451 0.29019608 0.29411765 0.29803922 0.30196078
 0.30588235 0.30980392 0.31372549 0.31764706 0.32156863 0.3254902
 0.32941176 0.33333333 0.3372549  0.34117647 0.34509804 0.34901961
 0.35294118 0.35686275 0.36078431 0.36470588 0.36862745 0.37254902
 0.37647059 0.38039216 0.38431373 0.38823529 0.39215686 0.39607843
 0.4        0.40392157 0.40784314 0.41176471 0.41568627 0.41960784
 0.42352941 0.42745098 0.43137255 0.43529412 0.43921569 0.44313725
 0.44705882 0.45098039 0.45490196 0.45882353 0.4627451  0.46666667
 0.47058824 0.4745098  0.47843137 0.48235294 0.48627451 0.49019608
 0.49411765 0.49803922 0.50196078 0.50588235 0.50980392 0.51372549
 0.51764706 0.52156863 0.5254902  0.52941176 0.53333333 0.5372549
 0.54117647 0.54509804 0.54901961 0.55294118 0.55686275 0.56078431
 0.56470588 0.56862745 0.57254902 0.57647059 0.58039216 0.58431373
 0.58823529 0.59215686 0.59607843 0.6        0.60392157 0.60784314
 0.61176471 0.61568627 0.61960784 0.62352941 0.62745098 0.63137255
 0.63529412 0.63921569 0.64313725 0.64705882 0.65098039 0.65490196
 0.65882353 0.6627451  0.66666667 0.67058824 0.6745098  0.67843137
 0.68235294 0.68627451 0.69019608 0.69411765 0.69803922 0.70196078
 0.70588235 0.70980392 0.71372549 0.71764706 0.72156863 0.7254902
 0.72941176 0.73333333 0.7372549  0.74117647 0.74509804 0.74901961
 0.75294118 0.75686275 0.76078431 0.76470588 0.76862745 0.77254902
 0.77647059 0.78039216 0.78431373 0.78823529 0.79215686 0.79607843
 0.8        0.80392157 0.80784314 0.81176471 0.81568627 0.81960784
 0.82352941 0.82745098 0.83137255 0.83529412 0.83921569 0.84313725
 0.84705882 0.85098039 0.85490196 0.85882353 0.8627451  0.86666667
 0.87058824 0.8745098  0.87843137 0.88235294 0.88627451 0.89019608
 0.89411765 0.89803922 0.90196078 0.90588235 0.90980392 0.91372549
 0.91764706 0.92156863 0.9254902  0.92941176 0.93333333 0.9372549
 0.94117647 0.94509804 0.94901961 0.95294118 0.95686275 0.96078431
 0.96470588 0.96862745 0.97254902 0.97647059 0.98039216 0.98431373
 0.98823529 0.99215686 0.99607843 1.        ]
[False  True]
```

Gambar 6.4 Hasil Normalisasi

6.3 Hasil Pelatihan Model

Gambar 6.5 Hasil pelatihan

Hasil pelatihan FCN-8s cukup baik dengan nilai secara umum metrik *accuracy*, *F1-score* dan *mIOU* cukup tinggi. Sampel gambar 6.5 menunjukkan model mampu memprediksi api yang besar maupun kecil dengan pencahayaan berbeda. Tabel 6.1 dan tabel 6.2 menguraikan hasil evaluasi training set. pada tingkat pixel. Evaluasi tingkat pixel memperlihatkan kelas setiap pixel dalam 1 citra berukuran 224x224 (50.176 pixel). Nilai elemen-elemen *confusion matrix* pada tabel 6.1 menunjukkan prediksi 11.339.776 pixel yang berasal dari jumlah total training set (226 citra) dikalikan dengan jumlah pixel dalam 1 citra (50.176 pixel).

Tabel 6.1 *Confusion matrix* training set tingkat pixel

N = 11.339.776	Positif ada api	Negatif tidak ada api
Prediksi ada api	*True Positive* 557.584	*False Positive* 105.104
Prediksi tidak ada api	*False Negative* 61.789	*True Negative* 10.615.259

Tabel 6.2 Metrik evaluasi training set tingkat pixel

Metrik	*Accuracy*	*F1-score*	*mIOU*
Nilai	98,52%	86,97%	76,95%

Selain dilakukan evaluasi pada tingkat pixel, evaluasi tingkat citra juga dilakukan untuk melihat hasil nyata pada deteksi api pada observasi langsung. Evaluasi tingkat citra dilakukan karena deteksi api pada system yang diusulkan bergantung pada observasi langsung hasil prediksi berupa citra yang dihasilkan oleh model, berbeda dengan klasifikasi *convolutional neural network* biasa yang hanya berupa kelas. Tabel 6.3 dan tabel 6.4 menguraikan hasil training set pada tingkat citra.

Tabel 6.3 *Confusion matrix* training set tingkat citra

N = 226	Positif ada api	Negatif tidak ada api
Prediksi ada api	*True Positive* 118	*False Positive* 10
Prediksi tidak ada api	*False Negative* 1	*True Negative* 97

Tabel 6.4 Metrik evaluasi training set tingkat citra

Metrik	*Accuracy*	*F1-score*
Nilai	95,13%	95,55%

Walaupun nilai dari *true positive* dan *true negative* cukup tinggi masih terdapat prediksi *false positive* pada 10 citra. Gambar 6.6 menunjukkan sampel terjadinya prediksi *false positive* pada model. Dari observasi dapat dilihat bahwa model masih salah memprediksi jika intensitas dari cahaya mirip apinya tinggi,

Gambar 6.6 Sampel false positive evaluasi training set

Metrik evaluasi training set tingkat citra tidak mencakup *mIOU* karena metrik *mIOU* diperuntukan khusus untuk similaritas antara *ground truth* dan hasil prediksi tingkat pixel. Jika dibandingkan dengan metrik akurasi pada tingkat pixel, evaluasi pada tingkat citra lebih baik.

Evaluasi test set

Hasil pelatihan dan training set tidak mencerminkan bagaimana performa model di situasi dunia nyata. Pada dasarnya pelatihan hanya mendeskripsikan performa model pada training set sehingga kualitas training set itu sendiri sangat berpengaruh pada performanya jika diuji dengan data set lain.

Gambar 6.7 Sampel prediksi test Set

Hasil prediksi pada test set pada tingkat pixel menurun drastic dibandingkan dengan training set. Tabel 6.6 menunjukkan perbandingan metrik akurasi test set disertai dengan training set untuk perbandingan. Metrik evaluasi menurun cukup drastis, tabel 6.6 menunjukkan metrik F1-score berkurang 29.92% dan *mIOU* berkurang 38.01% kecuali penurunan pada metrik *accuracy* yang hanya turun 2.41%.

Tabel 6.5 *Confusion matrix* test set tingkat pixel

N = 8.981.504	Positif ada api	Negative tidak ada api
Prediksi ada api	*True Positive* 222.573	*False Positive* 201.860
Prediksi tidak ada api	*False Negative* 147.101	*True Negative* 8.409.970

Tabel 6.6 Metrik akurasi test set tingkat pixel

Dataset	*Accuracy*	*F1-score*	*mIOU*
Test set	96,11%	56,05%	38,94%
Training set	98,52%	86,97%	76,95%

Hasil dari evaluasi terhadap test set juga mengalami penurunan yang signifikan pada tingkat citra. Ditemukan 31 citra terprediksi sebagai false positive, jumlah yang tinggi jika dibandingkan 10 citra pada training set. Kenaikan jumlah false positive jika kita bandingkan menggunakan metrik *false positive rate* menunjukkan kenaikan dari 9.34% menjadi 37.34%

Tabel 6.7 *Confusion matrix* test set tingkat citra

N=179	Positif ada api	Negative tidak ada api
Prediksi ada api	*True Positive* 88	*False Positive* 31
Prediksi tidak ada api	*False Negative* 9	*True Negative* 52

Tabel 6.8 Metrik evaluasi test set tingkat citra

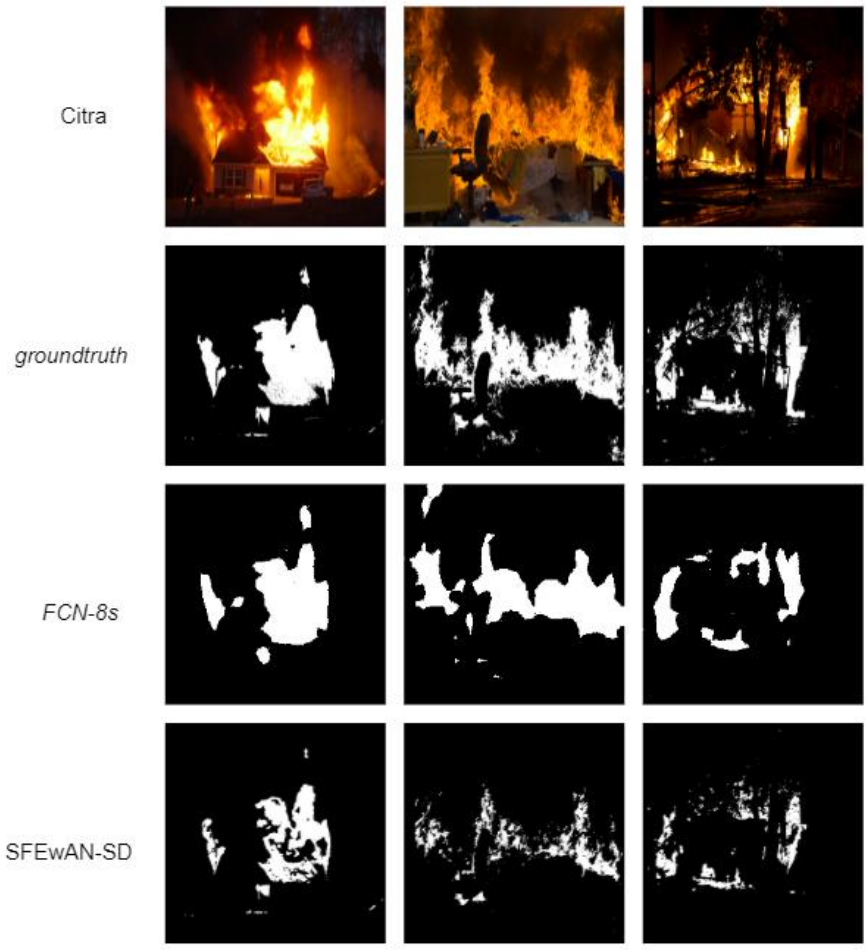
Dataset	*Accuracy*	*F1-score*
Test set	77,78%	81,48%
Training set	95,13%	95,55%

Hasil prediksi dari test set juga menunjukan hasil false negative yang sebelumnya tidak banyak ditemui pada training set. *False negative* ditemukan pada 9 citra dengan kondisi pencahyaan gelap maupun terang dan bentuk api besar maupun kecil

Gambar 6.8 Hasil false negative test set

Pada tingkat citra , sama halnya seperti evaluasi pada tingkat pixel metrik mengalami penurunan signifikan. Bisa dikatakan bahwa berarti training set tidak merepresntasikan kondisi nyata. Namun penurunan yang terjadi baik di tingkat pixel maupun tingkat citra masih melebihi 60%, tidak buruk untuk pelathian yang menggunakan training set yang bisa dibilang tidak banyak.

6.4 Perbandingan dengan SFEwAN-SD



Gambar 6.9 Perbandingan Prediksi FCN-8s dan SFEwAN-SD

Pelatihan SFEwAN-SD menggunakan training set yang sama dengan training set FCN-8s. Namun seperti dijelaskan pada bagian analisis dataset di Bab 4 setiap arsitektur aslinya mempunyai dimensi yang berbeda-beda. FCN-8s menggunakan training set yang dimensinya diubah menjadi 224x224 sedangkan SFEwAN-SD menggunakan training set yang diubah dimensinya menjadi 640x480.

Untuk memudahkan perbandingan secara observasi langsung maka Gambar 6.9 memuat sampel perbandingan prediksi pada test set dengan dimensi yang diseragamkan setiap citra menjadi 150x150. Dari gambar dapat dilihat bahwa FCN-8s menciptakan prediksi yang lebih kasar dalam bentuk *glob(*gumpalan) namun memuat setiap bagian api dalam citra. SFEwAN-SD menciptakan prediksi yang lebih detail pada pinggiran objek api namun seringkali tidak mendeteksi api yang sekiranya berbentuk besar seperti gambar tengah.

Hasil komparasi dengan SFEwAN-SD pada tabel 6.9 menunjukkan FCN-8s mengalahkan SFEwAN-SD pada F1-score dan mIOU pada tingkat pixel dengan margin yang signifikan, perbedaan 11.79% pada F1-score dan 14.9% pada mIOU. Pada *accuracy* FCN-8s kalah namun marginnya tidak signifikan pada 0.12%.

Tabel 6.9 Perbandingan metrik evaluasi tingkat pixel FCN-8s dan SFEwAN-SD

Model	*Accuracy*	*F1-Score*	*mIOU*
FCN-8s	96,11%	**50,56%**	**38,94%**
SFEwAN-SD	**96,23%**	38,77%	24,04%

Tabel 6.10 Perbandingan metrik evaluasi tingkat citra FCN-8s dan SFewAN-SD

Model	*Accuracy*	*F1-Score*
FCN-8s	**78,21%**	**81,86%**
SFEwAN-SD	57,54%	71,21%

6.5 Hasil Modifikasi SFEwAN-SD

Gambar 6.10 Perbandingan penambahan *skip connection*

Modifikasi SFEwAN-SD dengan menambahkan skip connection melalui *max pooling layer* diuji pada 3 layer max pooling yang digunakan: *Max Pool* 1.1, *Max Pool* 1.2, dan *Max Pool* 1.5. Setiap skip connection terdiri dari tiga layer: *Convolutional Layer*, *Transpose Convolutional Layer* dan *Cropping Layer*. *Cropping Layer* digunakan karena hasil transpose tidak dapat disamakan dengan resolusi 640x480 untuk digabungkan dengan layer terakhir dari model utama. Baik dari penurunan dan peningkatann yang terjadi cukup signifikan. Tabel 6.11 dan tabel 6.12 menunjukkan metrik evaluasi ketika diuji menggunakan test set.

Tabel 6.11 Perbandingan penambahan *skip connection* tingkat pixel

Model	*Accuracy*	*F1-Score*	*mIOU*
SFEwAN-SD	**96,23%**	38,77%	24,04%
SFEwAN-SD + *Skip connection* Max Pool 1.1	96,04%	16,51%	8,99%
SFEwAN-SD + *Skip connection* Max Pool 1.2	95,99%	12,77%	6,82%
SFEwAN-SD + *Skip connection* Max Pool 1.5	95,57%	**56,94%**	**39,8%**

Pada tabel 6.11 bisa dilihat bahwa penambahan *skip connection* sangat variative, skip connection melalui Max Pool 1.1 dan Max Pool 1.2 menunjukkan penurunan drastic untuk *F1-score* dan *mIOU*. Max Pool 1.5 sebaliknya menunjukkan peningkatan signifikan baik pada F1-score maupun mIOU. Untuk metrik accuracy sendiri setiap penambahan skip connection menunjukkan penurunan akurasi namun tidak signifikan.

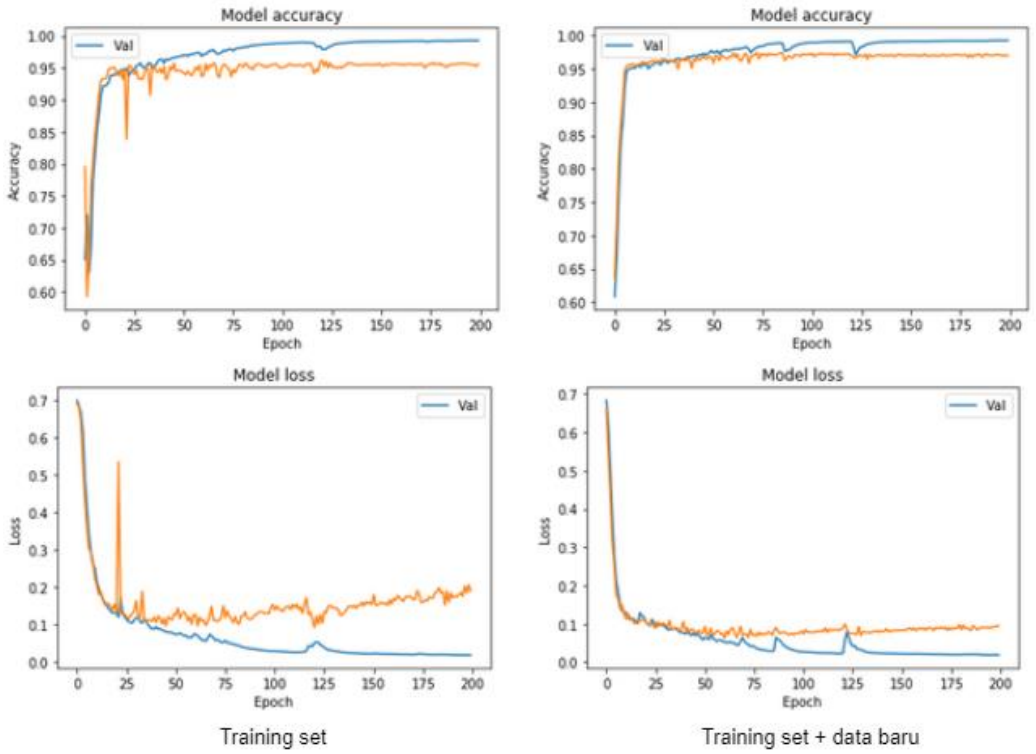
Tabel 6.12 Perbandingan penambahan *skip connection* tingkat citra

Model	*Accuracy*	*F1-Score*
SFEwAN-SD	57,54%	71,21%
SFEwAN-SD + *Skip connection Max Pool* 1.1	57,54%	66,67%
SFEwAN-SD + *Skip connection Max Pool* 1.2	60,34%	63,96%
SFEwAN-SD + *Skip connection Max Pool* 1.5	**60,89%**	**73,08%**

Pada evalusi tingkat citra hasil yang didapatkan juga tidak berbeda dengan hasil pada tingkat pixel. Penambahan skip connection *Max Pool* 1.1 dan *Max Pool* 1.2 menunjukkan penurunan atau stagnan sedangkan skip connection *max pool* 1.5 mengalami peningkatan.

Penurunan hasil akurasi model yang terjadi pada skip connection yang berasal dari *Max Pool* 1.1 dan *Max Pool* 1.2 dapat terjadi dikarenakan hasil operasi transpose convolution dan cropping tidak optimal sehingga hasil dari layer tersebut ketika digabungkan dengan layer dari model utama tidak menghasilkan bagian dengan objek api yang seragam.

6.6 Hasil Perubahan Jumlah Data Training Set



Gambar 6.11 Perbandingan *Accuracy* dan *Loss*

Untuk melihat signifikansi jumlah data yang digunakan pada akurasi model maka ditambahkan 204 citra baru pada training set yang ada. Total data yang digunakan pada pelatihan ini adalah 430 citra. Komposisi pelatihan kali ini membagi dataset menjadi 344 citra sebagai training set dan 86 citra sebagai validation set.

Gambar 6.11 menunjukkan grafik metrik *accuracy* dan loss saat pelatihan dilangsungkan dengan garis biru sebagai training set dan garis oranye sebagai validation set. Perbedaan margin yang membesar antara kedua garis tersebut dapat menunjukkan fenomena *overfitting* Pada training set lama baik dari sisi *accuracy*

dan *loss* seiring berjalannya epoch nilai dari validation set cenderung menjauhi training set, walaupun tidak menjauh secara signifikan namun pada *loss* pergerakannya cukup terlihat. Sedangkat pada training set baru pergerakan beda jarak antara *accuracy* dan *loss* kedua belah garis tersebut tidak sejauh training set lama. Dengan adanya komparasi tersebut bisa dikatakan bahwa penambahan dataset berkontribusi positif untuk performa model.

Metrik evaluasi untuk tingkat pixel di tabel 6.13 dan tabel 6.14 juga menunjukkan bukti bahwa training set baru menunjukkan kenaikan dari semua metrik walaupun hanya pada kisaran 1%

Tabel 6.13 *Confusion matrix* training set baru tingkat pixel

N=21.575.860	Positif ada api	Negatif tidak ada api
Prediksi ada api	*True Positive* 925.400	*False Positive* 150.840
Prediksi tidak ada api	*False Negative* 115.654	*True Negative* 20.383.786

Tabel 6.14 Metrik evaluasi tingkat pixel training set baru

Dataset	*Accuracy*	*F1-score*	*mIOU*
Training set + data baru	**98,76%**	**87,41%**	**77,64%**
Training set	98,52%	86,97%	76,95%

Sebaliknya pada tingkat citra, ditunjukkan pada tabel 6.15 dan tabel 6.16 semua metrik menurun pada kisaran 1%. Hal ini bisa dikarenakan walaupun jumlah *true positive* naik namun begitu juga jumlah *false positive* dan *false negative*. Bila kita lihat *false negative rate,*di mana nilainya adalah jumlah *false negative* banding jumlah total *true positive* dan *false negative,* training set lama mempunyai nilai 0.85%, namun training set baru 1.82% . Begitu juga dengan *false positive rate*, training set lama mempunyai nilai 9.34% dan training set baru mempunyai nilai 10.14%.

Tabel 6.15 *Confusion matrix* training set baru tingkat citra

N = 430	Positif ada api	Negatif tidak ada api
Prediksi ada api	*True Positive* 219	*False Positive* 21
Prediksi tidak ada api	*False Negative* 4	*True Negative* 186

Tabel 6.16 Metrik evaluasi training set baru tingkat citra

Dataset	*Accuracy*	*F1-score*
Training set + data baru	94,19%	94,60%
Training set	95,13%	95,55%

Evaluasi test set

Hasil evaluasi pada test set juga melihat kenaikan pada akurasi model setelah dilakukan pelatihan menggunakan dataset baru. Tabel 6.15 dan tabel 6.16 menunjukkan *confusion matrix* tingkat pixel dan tingkat citra pada pelatihan baru. Pada tingkat pixel jika dibandingkan dengan hasil evaluasi test set tsebelumnya hasil *true positive* dan *true negative* meningkat sedangkan *false negative* dan *false positive* menurun.

Tabel 6.17 *Confusion matrix* test set tingkat pixel untuk training set baru

N = 8.981.504	Positif ada api	Negative tidak ada api
Prediksi ada api	*True Positive* 227.594	*False Positive* 115.165
Prediksi tidak ada api	*False Negative* 142.080	*True Negative* 8.496.665

Tabel 6.18 *Confusion matrix* test set tingkat citra untuk training set baru

N=179	Positif ada api	Negative tidak ada api
Prediksi ada api	*True Positive* 86	*False Positive* 28
Prediksi tidak ada api	*False Negative* 10	*True Negative* 55

Perbandingan hasil test set menunjukkan bahwa semakin jumlah data berkorelasi dengan naiknya tingkat akurasi model. Pada tingkat citra hasil evaluasi menunjukkan peningkatan yang variatif pada metrik. Hasil peningkatan berkisar antara 1% sampai dengan 14%

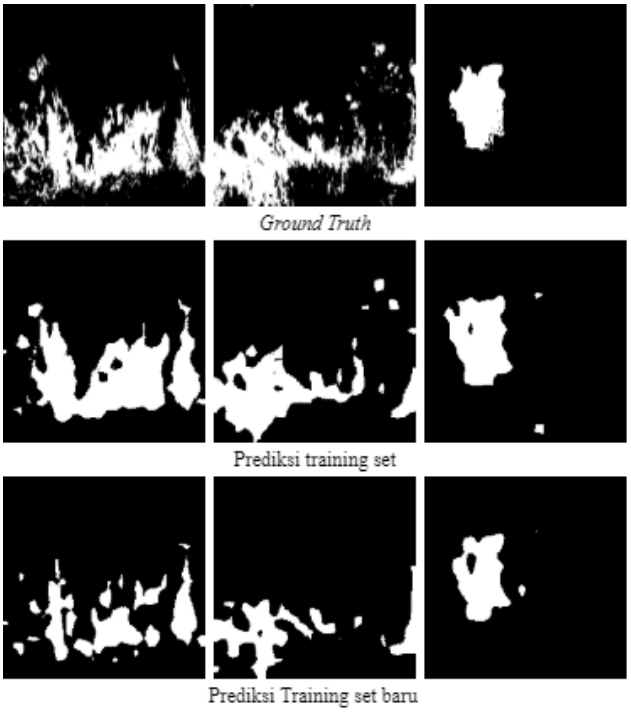
Tabel 6.19 Perbandingan evaluasi tingkat pixel

Model	*Accuracy*	*F1-Score*	*mIOU*
Pelatihan Training Set	96,11%	50,56%	38,94%
Pelatihan Training set+data baru	**97,13%**	**63,89%**	**46,94%**

Tabel 6.20 Perbandingan evaluasi tingkat citra

Model	*Accuracy*	*F1-Score*
Pelatihan Training Set	78,21%	81,86%
Pelatihan Training set+data baru	**78,77%**	**81,90%**

Hasil pada tabel 6.19 dan tabel 6.20 konsisten memperlihatkan kenaikan performa pada model namun peningkatan pada tingkat citra sangat tidak signifikan dengan nilai dibawah 1%. Perbedaan pada F1-score khususnya dengan kenaikan pada 13,23% pada tingkat pixel namun hanya 0,04% pada tingkat citra. Secara intuitif seharusnya evaluasi tingkat citra mengalami peningkatan signifikan.



Gambar 6.12 Perbandingan hasil prediksi training set baru

Gambar 6.12 memperlihatkan sampel perbandingan antara *ground truth,*pelatihan *training set* dan pelatihan *training set* baru. Dapat dilihat dari hasil pelatihan training set baru objek api tidak sebesar pada training set lama. Hasil training set baru lebih mendekati bentuk *ground truth* sehingga dapat disimpulkan bahwa pelatihan dengan training set baru mengurangi nilai-nilai false positive pada tingkat pixel.

Hasil observasi dari seluruh prediksi test set dengan training set menunjukkan bahwa penambahan training set lebih banyak berkontribusi pada pengurangan false positive dan false negative pada prediksi tingkat pixel. Pada tingkat citra pengurangan pada false positive dan false negative tingkat pixel tidak mempengaruhi atau mencerminkan penambahan true positive pada tingkat citra.

BAB VII
KESIMPULAN DAN SARAN

7.1 Kesimpulan

Hasil dari penelitian adalah sebagai berikut:

1. Arsitektur FCN-8s yang diusulkan mampu mendeteksi citra api secara akurat pada tingkat citra.
2. Pada tingkat pixel masih banyak objek api yang terlihat lebih besar daripada *groundtruth* atau dengan kata lain masih terdeteksi *false positive*.
3. Dari observasi langsung dapat dilihat bahwa FCN-8s lebih akurat saat mendeteksi api dalam skala besar.
4. Untuk mengenali citra bukan api arsitektur dapat mengenali objek bukan api secara baik namun masih tidak akurat ketika memprediksi citra yang mempunyai ciri khas sangat mirip api seperti lampion listrik dan lampu.
5. FCN-8s mengalahkan SFEwAN-SD pada tingkat citra untuk semua metrik evaluasi.
6. Turunnya metrik evaluasi pada test set menunjukkan bahwa hasil pelatihan menggunakan training set tidak sepenuhnya akurat untuk test set yang mempunyai jenis citra api dalam kondisi yang berbeda jauh dengan yang ada dalam training set.
7. Hasil penambahan data baru pada *training set* meningkatkan hasil akurasi model.

7.2 Saran

1. Penelitian berbasis segmentasi semantic untuk citra api belum terlalu banyak. Penelitian kedepan dapat menggunakan arsitektur lain untuk segmentasi semantic sepert Unet dan Resnet.
2. Dataset untuk segmentasi untuk masalah segmentasi api masih susah untuk didapatkan dan sedikit jumlahnya. Pada penelitian di masa yang akan datang akan lebih baik jika ada dataset yang mempunyai jumlah data yang lebih banyak dan contoh terjadinya api yang lebih variatif dalam aspek pencahayaan, kondisi dan lingkungan untuk menjadi standard pada deteksi api.

DAFTAR PUSTAKA

Bishop, C.M., 2006, *Machine Learning and Pattern Recognition* ,Springer, New York.

Ballard, D.H, dan Brown, C.M., 1982, *Computer Vision*, Prentice Hall, New Jersey.

Dimitropoulos, K., Barmpoutis, P., dan Grammalidis, N., 2015, Spatio-Temporal Flame Modeling and Dynamic Texture Analysis for Automatic Video-Based Fire Detection, *IEEE Transactions on Circuits and Systems for Video Technology*, 25, 2, 339-351.

Fausette, L., 1994, *Fundamental of Neural Network*, Prentice Hall, New Jersey.

Frizzi, S., Kaabi, R., Bouchouicha, M., Ginoux, J., Moreau, E., dan Fnaiech, F., 2016, Convolutional neural network for video fire and smoke detection, *Industrial Electronics Society , IECON 2016 - 42nd Annual Conference of the IEEE*,23-26 Oktober 2016.

Foggia, P., Saggese, A., dan Vento, M., 2015, Real-Time Fire Detection for Video-Surveillance Applications Using a Combination of Experts Based on Color, Shape, and Motion, *IEEE Transactions on Circuits and Systems for Video Technology*, 25, 9, 1545-1556.

Fukunaga, K., dan Hostetler, L., 1975, The estimation of the gradient of a density function, with applications in pattern recognition, *IEEE Transactions on Information Theory*, 21, 2, 32-40.

Goodfellow, I., Bengio, Y., dan Courville, A.,2017, *Deep Learning*, MIT Press, Cambridge.

Gonzalez, A., Zuniga, M.D., Nikulin, C., Carvajal, G., Cardenas, D.G, et al, 2017, Accurate Fire Detection through Fully Convolutional Network, *7th Latin American Conference on Networked and Electronic Media*, 6-7 Oktober 2017.

Khan, R.A., Uddin, J., Corraya, S., 2017, Real-Time Fire Detection Using Enhanced Color Segmentation and Novel Foreground Extraction, *Proceedings of the 2017 4th International Conference on Advances in Electrical Engineering (ICAEE)*,28-30 September 2017

Ko, B.C., Ham, S.J., dan Nam, J.Y., 2011, Modeling and Formalization of Fuzzy Finite Automata for Detection of Irregular Fire Flames, *IEEE Transactions on Circuits and Systems for Video Technology*, 21, 12, 1903-1912.

Krizhevsky, A., Sutskever, I., Hinton, G.E, 2012, ImageNet Classification with Deep Convolutional Neural Networks, *Proceedings of advances in Neural Information Processing Systems*, 3-6 Desember 2012, 1097-1105.

Muhammad, K., Ahmad, J., Mehmood, I., , Rho, S., dan Baik, S.W., 2018, Convolutional Neural Networks Based Fire Detection in Surveillance Videos, *IEEE Access*, 6,18174 - 18183.

Muhammad, K., Ahmad, J., dan Baik, S.W., 2017, Early fire detection using convolutional neural networks during surveillance for effective disaster management, *Neurocomputing*, 288,30–42.

Simonyan, K., dan Zisserman, A., 2014, Very Deep Convolutional Networks for Large-Scale Image Recognition, arXiv:1409.1556v6.

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., et al, 2015, Going Deeper with convolutions, *Proceedings of the IEEE COnference on Computer Science and Pattern Recognition*, 7-12 Juni 2015, 1-9.

Polednik, Bc. T., 2015, Detection of Fire in Images and Video, *Faculty of Information Technology, Brno University of Technology*.

Szeliski, R., 2010, *Computer Vision : Algorithms and Applications*, Springer, New York.

Toreyin, B., Dedeoglu, Y., Gudukbay, U., dan Cetin, A., 2005, Flame detection in video using hidden Markov models, *IEEE International conference on Image Processing*, 14 September 2005, 1230-1233.