**DETEKSI UBUR-UBUR BERBASIS VIDEO DENGAN METODE DEEP LEARNING**

****

**TUGAS AKHIR**

*Disusun dalam rangka memenuhi salah satu persyaratan*

*Untuk menyelesaikan program Strata-1 Departemen Teknik Informatika*

*Fakultas Teknik Universitas Hasanuddin*

*Makassar*

**Disusun Oleh:**

**FUAD KHAIRI HAMID**

**D421 15 014**

**DEPARTEMEN TEKNIK INFORMATIKA**

**FAKULTAS TEKNIK**

**UNIVERSITAS HASANUDDIN**

**MAKASSAR**

**2019**

**KATA PENGANTAR**

Puji dan syukur penulis panjatkan kepada Tuhan Yang Maha Esa karena berkat rahmat dan karunia-Nya sehingga tugas akhir yang berjudul “DETEKSI UBUR-UBUR BERBASIS VIDEO DENGAN DEEP LEARNING” ini dapat diselesaikan sebagai salah satu syarat dalam menyelesaikan jenjang Strata-1 pada Departemen Teknik Informatika Fakultas Teknik Universitas Hasanuddin.

Penulis menyadari bahwa dalam penyusunan dan penulisan laporan tugas akhir ini tidak lepas dari bantuan, bimbingan serta dukungan dari berbagai pihak, dari masa perkuliahan sampai dengan masa penyusunan tugas akhir. Oleh karena itu, penulis dengan senang hati menyampaikan terima kasih kepada:

1. Kedua Orang tua penulis, Bapak Abdul Hamid, S.Pd dan Ibu Syamsiah Salama, S.Pd yang selalu memberikan dukungan, doa, dan semangat serta selalu sabar dalam mendidik penulis sejak kecil;
2. Bapak Dr. Indrabayu S.T., M.T., M.Bus.Sys., selaku pembimbing I dan Ibu Dr. Ir. Ingrid Nurtanio, M.T., selaku pembimbing II yang selalu menyediakan waktu, tenaga, pikiran dan perhatian yang luar biasa untuk mengarahkan penulis dalam penyusunan tugas akhir;
3. Ibu Dr. Eng. Intan Sari Areni, S.T., M.T., dan Ibu Anugrahyani Bustamin., S.T., M.T., yang senantiasa memberikan nasehat, masukan, serta perhatian yang luar biasa kepada penulis dalam penyusunan tugas akhir;
4. Bapak Dr. Amil Ahmad Ilham, ST., M.IT., selaku Ketua Departemen Teknik Informatika Fakultas Teknik Universitas Hasanuddin atas bimbingannya selama masa perkuliahan penulis;
5. Para sahabat, teman-teman dan kakak-kakak AIMP *Research Group* FT UH yang telah memberikan begitu banyak bantuan selama penelitian, pengambilan data dan diskusi *progress* penyusunan tugas akhir;
6. Para kru Sapulidi yang senantiasa mendukung, memberikan masukan, dan nasehat dalam kehidupan perkuliahan selama ini;
7. Teman-teman Hypervisor FT UH atas dukungan dan semangat yang diberikan selama ini;
8. Segenap Staf dan Dosen Departemen Teknik Informatika Fakultas Teknik Universitas Hasanuddin yang telah membantu penulis.
9. Orang-orang berpengaruh lainnya yang tanpa sadar telah menjadi inspirasi penulis.

Akhir kata, penulis berharap semoga Allah SWT. berkenan membalas segala kebaikan dari semua pihak yang telah banyak membantu. Semoga Tugas Akhir ini dapat memberikan manfaat bagi pengembangan ilmu. Aamiin.

Wassalam

Makassar, September 2019

**ABSTRAK**

Ubur-ubur merupakan makhluk hidup dengan jumlah paling banyak di lautan. Peningkatan jumlah ubur-ubur ini dipengaruhi oleh pemanasan global, dimana meningkatnya suhu air laut menyebabkan proses perkembangbiakan ubur-ubur semakin meningkat. Keberadaan ubur-ubur ini berbahaya apabila bersentuhan dengan kulit manusia, karena dapat memberikan sengatan yang memicu reaksi alergi berat. Hasil riset menunjukkan bahwa ada sekitar 150 juta laporan kasus sengatan ubur-ubur setiap tahunnya di seluruh dunia. Di Indonesia sendiri kasus sengatan ubur-ubur sering terjadi, tercatat pada tahun 2019 ada ratusan kasus sengatan ubur-ubur yang dilaporkan, dengan tingkat resiko yang berbeda-beda. Untuk mencegah terjadinya sengatan ubur-ubur pada manusia secara dini, dibutuhkan sebuah sistem yang dapat mendeteksi secara dini keberadaan ubur-ubur tersebut. Deep Learning merupakan salah satu solusi untuk menyelesaikan berbagai permasalahan, salah satunya pendeteksian objek. Dengan memanfaatkan teknologi deep learning, dapat dibuat sebuah sistem yang dapat mendeteksi keberadaan ubur-ubur tersebut berikut dengan jenis-jenisnya. Data yang digunakan dalam penelitian ini sebanyak 100 citra, yang terbagi atas 4 jenis ubur-ubur. Data ini kemudian dibagi menjadi 80 data latih dan 20 data uji. Sedangkan algoritma yang digunakan adalah algoritma *You Only Look Once* (YOLO) generasi ke-3 untuk pendeteksian objek. Pada penelitian ini diperoleh akurasi sistem sebesar 99.34%.

**Kata kunci:** ubur-ubur,*machine learning,* *deep learning, you only look once, neural network, object detection.*

**DAFTAR ISI**

[KATA PENGANTAR i](#_Toc19091769)

[ABSTRAK iii](#_Toc19091770)

[DAFTAR ISI iv](#_Toc19091771)

[DAFTAR GAMBAR vi](#_Toc19091772)

[DAFTAR TABEL viii](#_Toc19091773)

[BAB I PENDAHULUAN 1](#_Toc19091774)

[1.1. Latar Belakang 1](#_Toc19091775)

[1.2. Rumusan Masalah 4](#_Toc19091776)

[1.3. Tujuan Penelitian 4](#_Toc19091777)

[1.4. Manfaat Penelitian 4](#_Toc19091778)

[1.5. Batasan Masalah Penelitian 5](#_Toc19091779)

[1.6. Metode Penulisan 5](#_Toc19091780)

[1.7. Sistematika Penulisan 6](#_Toc19091781)

[BAB II TINJAUAN PUSTAKA 7](#_Toc19091782)

[2.1. Ubur-ubur 7](#_Toc19091783)

[2.2. Pengolahan Citra Digital 8](#_Toc19091784)

[2.3. Visi Komputer 21](#_Toc19091785)

[2.4. Jaringan Saraf Tiruan 27](#_Toc19091786)

[2.5. Activation Function 30](#_Toc19091787)

[2.1.6. Algoritma You Only Look Once v3 (YOLO) 33](#_Toc19091788)

[2.6. Penelitian Terkait 34](#_Toc19091789)

[BAB III METODOLOGI PENELITIAN 38](#_Toc19091790)

[3.1. Tahapan Penelitian 38](#_Toc19091791)

[3.2. Waktu dan Lokasi Penelitian 39](#_Toc19091792)

[3.3. Instrumen Penelitian 40](#_Toc19091793)

[3.4. Teknik Pengambilan Data 40](#_Toc19091794)

[3.5. Perancangan Sistem 41](#_Toc19091795)

[3.6. Analisis Kerja Sistem 49](#_Toc19091796)

[BAB IV HASIL DAN PEMBAHASAN 51](#_Toc19091797)

[4.1. Hasil Penelitian 51](#_Toc19091798)

[4.2. Pembahasan 53](#_Toc19091799)

[BAB V PENUTUP 56](#_Toc19091800)

[5.1. Kesimpulan 56](#_Toc19091801)

[5.2. Saran 57](#_Toc19091802)

[DAFTAR PUSTAKA 58](#_Toc19091803)

[LAMPIRAN 60](#_Toc19091804)

# DAFTAR GAMBAR

Gambar 2. 1 Ubur-ubur 7

Gambar 2. 2 Contoh Matriks Citra RGB 12

Gambar 2. 3 Hasil Konversi dengan Lightness Method 13

Gambar 2. 4 Hasil Konversi dengan Average Method 14

Gambar 2. 5 Hasil Konversi dengan Luminosity Method 15

Gambar 2. 6 Jenis-jenis Tepi 17

Gambar 2. 7 Perubahan Citra Sebelum dan Sesudah DT 18

Gambar 2. 8 Matriks Perubahan Citra Sebelum dan Sesudah DT 18

Gambar 2. 9 Contoh Operasi Hit pada DIlasi 19

Gambar 2. 10 Contoh Operasi Fit pada Erosi 20

Gambar 2. 11 *Perbandingan Hasil Ekstraksi Blob* 21

Gambar 2. 12 *Ekualisasi Histogram* 23

Gambar 2. 13 Citra *Threshold* 26

Gambar 2. 14 *Single Layer Network* 29

Gambar 2. 15 *Multi Layer Network* 30

Gambar 2. 16 Linear Function 30

Gambar 2. 17 Non Linear Activation Function 31

Gambar 2. 18 Rectified Linear Unit 32

Gambar 2. 19 Perbandingan Performa YOLO 34

Gambar 3. 1 Diagram Tahapan Penelitian 38

Gambar 3. 2 Rancangan Sistem Deteksi Ubur-ubur 41

Gambar 3. 3 Contoh Jenis Ubur-ubur 42

Gambar 3. 4 Ilustrasi Proses *Resize* Citra 43

Gambar 3. 5 Convolution Layer 44

Gambar 3. 6 Max Pooling Layer 44

Gambar 3. 7 Proses *convolution* dan *max pooling* pada citra 45

Gambar 3. 8 Proses *convolution* dan *max pooling* pada *layer* kedua 45

Gambar 3. 9 *Activation map* dari proses *convolution* dan *max pooling* 46

Gambar 3. 10 Proses *reshape activation map* menjadi dari *multidimensional array* menjadi vektor 46

Gambar 3. 11 *Fully Connected Layer* pada Sistem 47

[Gambar 3. 12 Output dari *Fully Connected Layer*](#_Toc15761048) 48

[Gambar 3. 13 Visualisasi Output dari Sistem](#_Toc15761048) 49

[Gambar 3. 14 Confusion Matrix 5](#_Toc15761049)2

[Gambar 4. 1 Rincian hasil training sistem dengan 10000 iterasi 5](#_Toc15761050)2

# DAFTAR TABEL

Tabel 2. 1 Penelitian Terkait 42

Tabel 4. 1 Confusion Matriks pelatihan 10000 iterasi 52

**BAB I  
PENDAHULUAN**

* 1. **Latar Belakang**

Indonesia merupakan negara dengan wilayah laut yang lebih luas dibanding wilayah daratannya sekaligus memiliki garis pantai terpanjang ke-2 di dunia. Hal ini membuat Indonesia memiliki keanekaragaman hayati di laut serta memiliki potensi menjadikan banyak daerah pantai sebagai objek wisata. Dengan keanekaragaman yang ada di lautan Indonesia, tentunya terdapat berbagai jenis makhluk hidup yang ada di laut yang berpotensi membahayakan bagi pengunjung objek wisata pantai. Berbagai makhluk hidup yang ada di daerah pantai seperti ubur-ubur dapat menyebabkan permasalahan serius jika tidak diindahkan keberadaannya.Ubur-ubur merupakan makhluk hidup yang jumlahnya sangat banyak di laut. Pada kedalaman 0-200 meter terdapat sekitar 38 juta ton ubur-ubur. Ubur-ubur ada di semua lautan termasuk pada lautan dalam, namun pada umumnya berada di daerah pesisir pantai (Lucas et al., 2014).

Jumlah ubur-ubur di laut terus bertambah dalam jumlah yang sangat besar, hal ini disebabkan oleh proses perkembangbiakannya yang dapat menghasilkan keturunan yang sangat banyak. Seekor ubur-ubur bulan betina dapat melepaskan lebih dari 400.000 bayi dalam satu kali proses berkembang biak. (Ishii, 2003)**.** Ubur-ubur dikategorikan sebagai makhluk hidup yang berbahaya karena mayoritas dari jenis ubur-ubur yang ada memiliki kemampuan untuk menyengat. Sengatan dari ubur-ubur dapat menyebabkan reaksi alergi berat yang dikenal dengan anafilaksis (Graham et al., 2014).

Hanya dalam hitungan menit setelah tersengat oleh ubur-ubur, sistem kekebalan tubuh akan bereaksi dan menyebabkan pembengkakan pada wajah, jantung berdebar, ruam dan gatal, susah bernapas, hingga dapat menyebabkan kematian pada manusia. Oleh sebab itu, keberadaan ubur-ubur yang muncul dalam jumlah banyak di pantai adalah hal yang membahayakan bagi pengunjung pantai tersebut. Bentuk ubur-ubur yang transparan juga menyulitkan manusia untuk melihat keberadaan ubur-ubur tersebut dari permukaan laut sehingga sulit untuk menghindari sengatannya (Graham et al., 2014).

Setiap tahunnya, diperkirakan ada sekitar 150 juta laporan kasus sengatan ubur-ubur pada pengunjung pantai di seluruh dunia.Keberadaan ubur-ubur di pantai biasanya dipengaruhi oleh perubahan musim. Perubahan musim yang terjadi menyebabkan perubahan arus laut hingga membawa koloni ubur-ubur ke daerah pantai. Selain karena pergantian musim, banyak factor lain yang dapat menyebabkan populasi ubur-ubur tiba-tiba meledak di suatu kawasan pantai. Misalnya terjadinya *blooming* populasi plankton atau algae beracun yang mengancam populasi makhluk hidup lain di laut lepas. Selain itu fenomena alam lainnya seperti gempa atau tsunami, gerhana matahari atau bulan juga dapat mempengaruhi arah migrasi dari biota laut, termasuk ubur-ubur.Berdasarkan latar belakang yang telah dijelaskan di atas, maka dalam penelitian ini dibuatlah sebuah sistem yang dapat mendeteksi keberadaan ubur-ubur menggunakan metode deep learning (*neural network*).

Sistem ini akan menganalisa video ubur-ubur kemudian menganalisa data video tersebut untuk mendeteksi keberadaan ubur-ubur.Terdapat beberapa penelitian yang telah menerapkan teknik *deep learning* untuk mendeteksi objek-objek bawah laut. Juhwal Kim dan Son-Cheul Yu pada tahun 2016 membuat suatu sistem yang dapat mendeteksi keberadaan ROV (*Remote Operated Vehicle*) yang mengeksplorasi lautan dalam dengan menggunakan inputan data berupa sonar-images. Metode yang digunakan adalah *deep learning* (*convolutional neural network*) dengan menggunakan arsitektur YOLO (*You Only Look Once*). Hasilnya adalah sistem tersebut dapat memantau pergerakan ROV secara lebih efektif menggunakan kombinasi *sonar-images* dan arsitektur YOLO.Kemudian untuk objek bulu babi dan timun laut, Danelle E. Cline dkk. melakukan penelitian dengan membuat suatu sistem pendeteksi dengan menggunakan iLab *Neumorphic C++ Vision toolkit* yang dikembangkan di *University of California*. Dengan menggunakan *toolkit* ini sistem yang dibangun mendapatkan akurasi 100% saat mendeteksi *benthic echinoderm rathbunaster californicus* (jenis bulu babi), dan 95% pada *parasthicopus leukothele* (jenis timun laut). Video bawah laut yang dianalisa merupakan video yang direkam menggunakan *deep sea* ROV.Untuk mendeteksi dan mengklasifikasi jenis-jenis ikan, penelitian yang dilakukan oleh Lin Meng dkk. menggunakan metode *convolutional neural network* (CNN) untuk membedakan jenis-jenis ikan pada video yang direkam menggunakan *underwater drone*. Penelitian ini menghasilkan sebuah sistem pendeteksi dan klasifikasi jenis-jenis ikan dengan akurasi mencapai 87%.

* 1. **Rumusan Masalah**

Rumusan masalah pada tugas akhir ini adalah:

1. Bagaimana cara membuat sistem untuk mendeteksi ubur-ubur berbasis video dengan menggunakan metode deep learning?
2. Bagaimana unjuk kerja sistem pendeteksi ubur-ubur berbasis metode deep learning?
   1. **Tujuan Penelitian**

Tujuan akhir dari penelitian ini adalah:

1. Untuk membuat sistem yang dapat mendeteksi ubur-ubur dengan akurat menggunakan metode deep learning.
2. Untuk mengetahui hasil unjuk kerja sistem untuk mendeteksi ubur-ubur dengan metode deep learning.
   1. **Manfaat Penelitian**

Dengan dilakukannya penelitian ini, diharapkan manfaat yang didapatkan adalah:

1. Bagi masyarakat, sistem yang dibuat dapat digunakan sebagai alat untuk mendeteksi keberadaan ubur-ubur, agar dapat terhindar dari bahaya sengatan ubur-ubur.
2. Bagi peneliti, penelitian ini dapat digunakan untuk menambah pengetahuan dan kemampuan di bidang *deep learning* untuk mendeteksi objek-objek yang berada di bawah laut.
3. Bagi institusi pendidikan, penelitian ini dapat digunakan sebagai referensi ilmiah untuk penelitian-penelitian terkait.
   1. **Batasan Masalah Penelitian**

Yang menjadi batasan masalah dalam tugas akhir ini adalah:

1. Objek penelitian adalah ubur-ubur.
2. Data yang digunakan untuk proses pelatihan dan pengujian diambil dari *dataset* *Open Images* v4. Sementara untuk unjuk kerja sistem menggunakan citra dan video dari berbagai sumber di Internet.
3. Rancangan model pendeteksi objek ubur-ubur dibuat menggunakan *deep learning* dengan metode *You Only Look Once* (YOLO) generasi ketiga.
   1. **Metode Penulisan**

Terdapat beberapa metode yang digunakan pada penelitian ini yaitu:

1. Metode Pengambilan Data

Metode pengambilan data dilakukan dengan mengumpulkan data dari sumber internet, antara lain data yang diekstraksi dari *Open Images Dataset v4* dan situs penyedia video seperti *Youtube*.

1. Studi Literatur

Studi literatur dilakukan dengan mengumpulkan informasi yang terkait dengan penelitian ini dari berbagai sumber seperti buku, internet, dan sumber lainnya.

1. Diskusi dan Konsultasi

Diskusi dan konsultasi dilakukan dengan mengadakan tanya jawab secara langsung kepada dosen pembimbing dan pihak-pihak profesional lainnya yang berhubungan dengan penelitian ini.

* 1. **Sistematika Penulisan**

Untuk memberikan gambaran singkat mengeni isi tulisan secara keseluruhan, maka akan diuraikan beberapa tahapan dari penulisan secara sistematis, yaitu :

BAB I PENDAHULUAN

Bab ini menguraikan secara umum mengenai hal yang menyangkit latar belakang, perumusan masalah dan baasan masalah, tujuan, mandaat, dan sistematika penulisan.

BAB II TINJAUAN PUSTAKA

Bab ini berisi teori-teori tentang hal-hal yang berhubungan dengan ubur-ubur, visi computer, pemrosesan citra, machine learning, deep learning, dan metode yang digunakan.

BAB III METODOLOGI PENELITIAN

Bab ini berisi tentang tahapan penelitian, instrument penelitian,dan penerapan algoritma serta teknik pengolahan data.

BAB IV HASIL DAN PEMBAHASAN

Bab ini berisi tentang hasil pengolahan data serta pembahasan yang disertai tabel hasil penelitian.

BAB V PENUTUP

Bab ini berisi tentang kesimpulan yang didapatkan berdasarkan hasil penelitian yang telah dilakukan serta saran-saran untuk pengembangan lebih lanjut.

**BAB II   
TINJAUAN PUSTAKA**

**2.1. Ubur-ubur**

Ubur-ubur atau *Scyphozoa* merupakan koelenterata yang hidup di laut baik dalam bentuk polip yang melekat di dasar ataupun yang berenang bebas dalam bentuk medusa. Tubuhnya lunak seperti gelatin, transparan, dan mengandung banyak air. Bentuk tubuhnya unik sehingga dengan mudah dapat dibedakan dari jenis koelenterata lainnya. Contoh jenis ubur-ubur dapat dilihat pada **Gambar 2.1.** berikut.



**Gambar 2.1.** Ubur-ubur jenis *Catostylus* (Medium, 2018).

Ubur-ubur ini dikenal sebagai binatang pengganggu di perairan dekat pantai terutama pada tempat-tempat rekreasi, karena dapat menyebabkan rasa gatal pada kulit bila tersentuh. Hal ini disebabkan oleh sel-sel penyengat atau nematosis yang terdapat di dalam jaringan epidermisnya, baik pada tentakel maupun di bagian lain tubuhnya. Dengan berkembangnya dunia perikanan, pengolahan binatang laut non ikan sudah banyak digalakkan. Ubur-ubur merupakan salah satu sumberdaya laut yang dapat diekspor, dan di lain pihak dapat menambah penghasilan kaum nelayan.

Di beberapa negara di Asia Tenggara, Jepang, Hongkong dan Korea, ubur-ubur telah dikenal sebagai salah satu bahan makanan bergizi karena mengandung kadar protein yang cukup tinggi. Di India terutama di daerah Tamil, ubur-ubur dari marga *Rhizostoma* dengan nama populernya *"Muttai Chori"* ditangkap dalam jumlah besar terutama pada bulan Januari sampai bulan Juni untuk diolah. Jenis-jenis dari marga ini diameter medusanya yang berbentuk payung dapat mencapai 80 cm.

**2.2. Pengolahan Citra Digital**

Pengolahan citra digital merupakan metode yang digunakan untuk mengolah citra pada komputer sehingga dapat menghasilkan gambar sesuai dengan yang dibutuhkan. Misalnya terdapat sebuah citra digital berwarna dengan ukuran 1280x720 piksel. Dengan pengolahan citra digital, gambar tersebut dapat diubah ukurannya menjadi 640x360 piksel tanpa mengurangi kualitas gambar. Citra merupakan fungsi kontinu dari intensitas cahaya dalam bidang dua dimensi. Secara matematis, fungsi ini dapat dilambangkan dengan *f(x,y),* dimana *(x,y)* merupakan koordinat pada bidang dua dimensi tersebut, dan *f(x,y)* merupakan intensitas cahaya pada titik *(x,y).* Ukuran terkecil dalam citra digital biasa disebut *picture element* atau *pixel* (Syawaluddin, 2016).

Citra bergerak (*moving images*) adalah rangkaian citra diam yang ditampilkan secara beruntun sehingga memberi kesan pada mata kita sebagai gambar yang bergerak. Setiap citra di dalam rangkaian itu disebut *frame*. Gambar-gambar yang tampak pada film layar lebar atau televisi pada hakikatnya terdiri dari ratusan hingga ribuan *frame*. Meskipun sebuah citra kaya informasi, namun seringkali citra mengalami penurunan mutu (degradasi), misalnya mengandung cacat atau derau, warnanya terlalu kontras, kurang tajam, dan kabur. Tentu saja citra semacam ini menjadi lebih sulit diinterpretasi karena informasi yang disampaikan oleh citra tersebut menjadi berkurang. Agar citra yang mengalami gangguan mudah diinterpretasikan (baik oleh manusia maupun mesin), maka citra tersebut perlu dimanipulasi menjadi citra lain yang kualitasnya lebih baik (Syawaluddin, 2016).

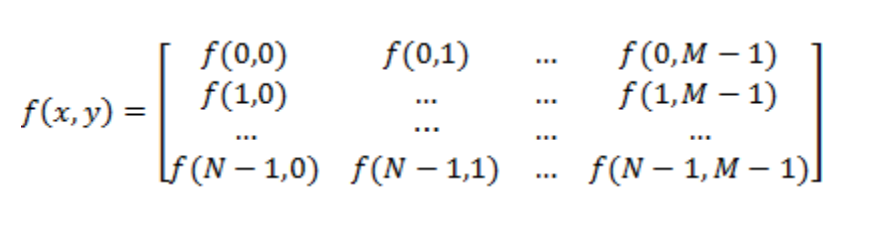
Di dalam mengolah sebuah citra, terdapat berbagai algoritma yang dapat diterapkan untuk menghasilkan keluaran yang lebih baik. Keluaran yang baik akan mempengaruhi hasil dari proses yang akan dilakukan selanjutnya.

1. **Citra Digital**

Citra merupakan kombinasi antara titik, garis, bidang, dan warna yang mewakili suatu objek atau benda. Sedangkan citra digital merupakan keluaran yang dihasilkan melalui perangkat pencitraan digital seperti kamera dan dapat disimpan oleh komputer digital.

Untuk menerjemahkan citra menjadi angka-angka yang dapat dipahami oleh komputer, maka citra dibagi menjadi bagian-bagian kecil yang disebut sebagai *pixel* (*picture elements*).

Dalam setiap *pixel*, perangkat pencitraan merekam sebuah angka ataupun sekumpulan angka yang merepresentasikan beberapa properti pada *pixel* yang dituju, seperti intensitas cahayanya maupun warnanya. Angka-angka ini diatur dalam sebuah baris-baris dan kolom-kolom *array* yang merujuk pada posisi vertikal dan horizontal dari *pixel*-*pixel* yang ada dalam citra. (Syawaluddin, 2016). Berikut adalah persamaan 2.1 yang menunjukkan representasi citra jika dibuat dalam bentuk matriks.



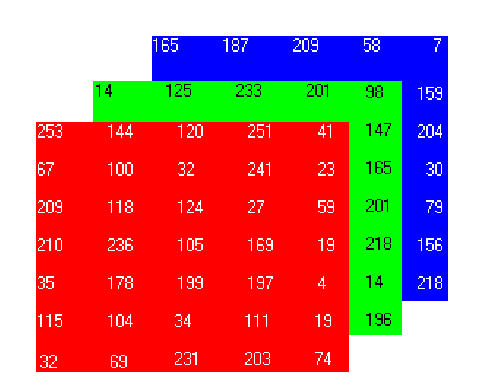
(2.1)

Menurut Darmawan dan Saptiani (2010), citra digital mengandung elemen-elemen dasar. yaitu:

1. Kecerahan (*Brightness*) merupakan intensitas cahaya rata-rata dari suatu area yang melingkupinya.
2. Kontras (*Contrast*) merupakan sebaran terang (*lightness*) dan gelap (*darkness*) di dalam sebuah citra. Citra dengan kontras rendah komposisi citranya sebagian besar terang atau sebagian besar gelap. Citra dengan kontras yang baik, komposisi gelap dan terangnya, tersebar merata.
3. Kontur (*Contour*) merupakan keadaan yang ditimbulkan oleh perubahan intensitas pada *pixel*-*pixel* tetangga, sehingga dapat dideteksi tepi objek di dalam citra.
4. Warna (*Color*) merupakan persepsi yang dirasakan oleh sistem visual manusia terhadap panjang gelombang cahaya (λ) yang dipantulkan oleh objek. Warna-warna yang dapat ditangkap oleh mata manusia merupakan kombinasi cahaya dengan panjang berbeda. Kombinasi yang memberikan rentang warna paling lebar adalah *red* (R), *green* (G) dan *blue* (B).
5. Bentuk (*Shape*) merupakan properti intrinsik dari objek tiga dimensi, dengan pengertian bahwa bentuk merupakan properti intrinsik utama untuk visual manusia. Umumnya citra yang dibentuk oleh manusia merupakan 2D, sedangkan objek yang dilihat adalah 3D.
6. Tekstur (*Texture*) merupakan distribusi spasial dari derajat keabuan di dalam sekumpulan *pixel*-*pixel* yang bertetangga.
7. **Citra Warna**

Pada citra warna, setiap *pixel* yang terdapat di dalamnya mewakili warna yang merupakan kombinasi dari tiga warna dasar yaitu merah (*red*), hijau (*green*), dan biru (*blue*)*.* Biasanya warna-warna dasar ini disebut sebagai *channel*. Setiap warna dasar menggunakan penyimpanan 8 bit = 1 *byte*, yang berarti setiap warna mempunyai gradasi 255 warna. Berarti setiap *pixel* mempunyai kombinasi warna sebanyak 28. 28. 28 = 224 = 16 juta warna lebih. Itu sebabnya format ini dinamakan *true color* karena mempunyai jumlah warna yang cukup besar sehingga bisa dikatakan hampir mencakup semua warna di alam.

Penyimpanan *true color* di dalam memori berbeda dengan citra *grayscale*. Setiap *pixel* dari citra *grayscale* yang terdiri atas 256 gradasi warna hanya diwakili oleh 1 *byte*. Sedangkan 1 *pixel* citra *true color* diwakili oleh 3 *byte*, di mana masing-masing *byte* merepresentasikan tiap R, G, dan B. **Gambar 2.2** berikut merupakan contoh representasi matriks dengan channel R, G, dan B.

**

**Gambar 2.2.** Contoh Matriks Citra RGB (Courtney, 2001).

1. Citra Grayscale

Sesuai dengan namanya, jenis citra ini memiliki gradasi warna hitam dan putih, yang menghasilkan efek warna abu-abu. Intensitas berkisar antara 0 sampai dengan 255. Nilai 0 menyatakan hitam dan 255 menyatakan putih. Citra RGB dengan matriks penyusun citra yang sebelumnya 3 matriks akan berubah menjadi 1 matriks saja.

Citra *grayscale* merupakan citra yang nilai intensitas *pixel*nya didasarkan pada derajat keabuan. Pada citra *grayscale* 8-bit, derajat warna hitam sampai dengan putih dibagi ke dalam 256 derajat keabuan dimana warna hitam sempurna direpresentasikan dengan nilai 0 dan putih sempurna dengan nilai 255. Citra RGB dapat dikonversi menjadi citra *grayscale* sehingga dihasilkan hanya satu kanal warna. Terdapat tiga metode yang digunakan untuk mengonversi citra RGB ke *grayscale*:

1. *Lightness Method*

Metode ini mengambil rata-rata pada intensitas warna yang paling mencolok dan paling tidak mencolok dengan persamaan 2.2 sebagai berikut:

|  |  |
| --- | --- |
|  | (2.2) |

Contoh hasil *grayscale* dengan *lightness method* dapat dilihat pada **Gambar 2.3** berikut.



**Gambar 2.3.**  Hasil Konversi dengan *Lightness Method* (Cook, 2009)

1. *Average Method*
2. *Average Method*

Metode rerata (*average method*) merupakan metode yang paling sederhana. Nilai *pixel* dari tiap-tiap *channel* diambil dan dirata-ratakan. Persamaan untuk *average method* dapat dilihat pada persamaan 2.3. Untuk contoh hasil konversi dengan *average method* dapat dilihat pada **Gambar 2.4**.

|  |  |
| --- | --- |
|  | (2.3) |
|  | | |  |

**

**Gambar 2.4.** Hasil Konversi dengan *Average Method* (Cook, 2009)

1. *Luminosity Method*

Metode ini memberikan nilai beban-beban tertentu pada *channel-channel* tertentu sesuai dengan kebutuhan pengguna. Manusia lebih sensitif terhadap warna hijau dibandingkan warna-warna lainnya, jadi warna biru biasanya memiliki bobot yang paling besar. Adapun salah satu contoh persamaan dari metode ini dapat dilihat pada persamaan 2.8, sementara hasil konversi dapat dilihat pada **Gambar 2.5** berikut:

|  |  |
| --- | --- |
|  | (2.4) |
|  |  |



**Gambar 2.5.** Hasil Konversi dengan *Luminosity Method* (Cook, 2009)

1. *Thresholding*

*Thresholding* merupakan salah satu teknik dasar dalam melakukan segmentasi citra. *Thresholding* digunakan untuk mengonversi citra *grayscale* menjadi citra biner. Citra biner hanya terdiri dari dua intensitas warna yaitu hitam yang memiliki representasi nilai 0 dan putih yang memiliki representasi nilai 1. Sehingga jenis citra in hanya membutuhkan 1 bit memori untuk menyimpan kedua warna ini.

*Thresholding* secara umum digunakan untuk memisahkan beberapa objek target dari latar belakangnya. Objek target biasanya direpresentasikan dengan warna putih. Adapun algoritma yang digunakan pada *global thresholding* ditunjukkan pada persamaan (2.5).

|  |  |
| --- | --- |
| Untuk semua *pixel* : | (2.5) |

Berdasarkan persamaan (2.5), jika nilai *pixel* pada koordinat lebih besar atau sama dengan nilai ambang, *T,* yang telah ditentukan sebelumnya, maka nilai tersebut akan diubah menjadi 1 dengan warna putih. Sedangkan jika nilai *pixel* pada koordinat lebih kecil dibandingkan dengan nilai ambang, *T,* maka nilai tersebut akan diubah menjadi 0 dengan warna hitam.

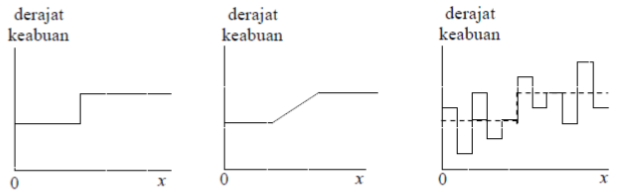
1. Deteksi Tepi

Faktor kunci dalam mengekstraksi ciri adalah kemampuan mendeteksi keberadaan tepi dari objek di dalam citra. Setelah tepi objek diketahui, langkah selanjutnya dalam analisis citra adalah segmentasi, yaitu mereduksi citra menjadi objek atau region, misalnya memisahkan objek-objek yang berbeda dengan mengekstraksi batas-batas objek. Langkah terakhir dari analisis citra adalah klasifikasi, yaitu memetakan segmen-segmen yang berbeda ke dalam kelas objek yang berbeda pula.

Ada tiga macam tepi yang terdapat di dalam citra digital. Ketiganya adalah:

1. Tepi curam memiliki perubahan intensitas yang tajam. Arah tepi berkisar 90°.
2. Tepi landai atau disebut juga tepi lebar, yaitu tepi dengan sudut arah yang kecil. Tepi landai dapat dianggap terdiri dari sejumlah tepi-tepi lokal yang lokasinya berdekatan.
3. Tepi yang mengandung derau umumnya memiliki tepi yang tidak konsisten.

Ilustrasi berbagai jenis tepi dapat dilihat pada **Gambar 2.6** berikut.



**Gambar 2.6.** Jenis-jenis Tepi: (a) Tepi Curam; (b) Tepi Landai; (c) Tepi Curam dengan Derau

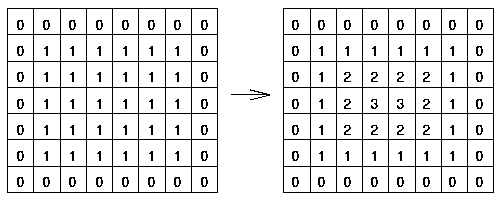
1. Distance Transform

*Distance transform* merupakan operator yang normalnya hanya diaplikasikan terhadap citra biner.Algoritma *distance transform* merupakan algoritma yang bagus untuk berbagai aplikasi seperti pengolahan citra, visi komputer, pengenalan pola, analisis bentuk, dan geometri komputasi (Arcelli dkk, 2011).

Citra yang dihasilkan oleh transformasi ini adalah citra *grayscale* yang terlihat mirip dengan citra masukannya, kecuali intensitas keabuan dari titik-titik di dalam daerah objek (*foreground*)diubah sedemikian rupa untuk menunjukkan jarak ke batas terdekat dari setiap titik seperti pada **Gambar 2.7**. Jika dilihat dalam bentuk matriks, maka penerapan *distance transform* terhadap citra biner, dapat dilihat pada **Gambar 2.8**.

**Gambar 2.7.** Perubahan Citra Sebelum dan Sesudah Penerapan Distance Transform (Fisher dkk, 2003)



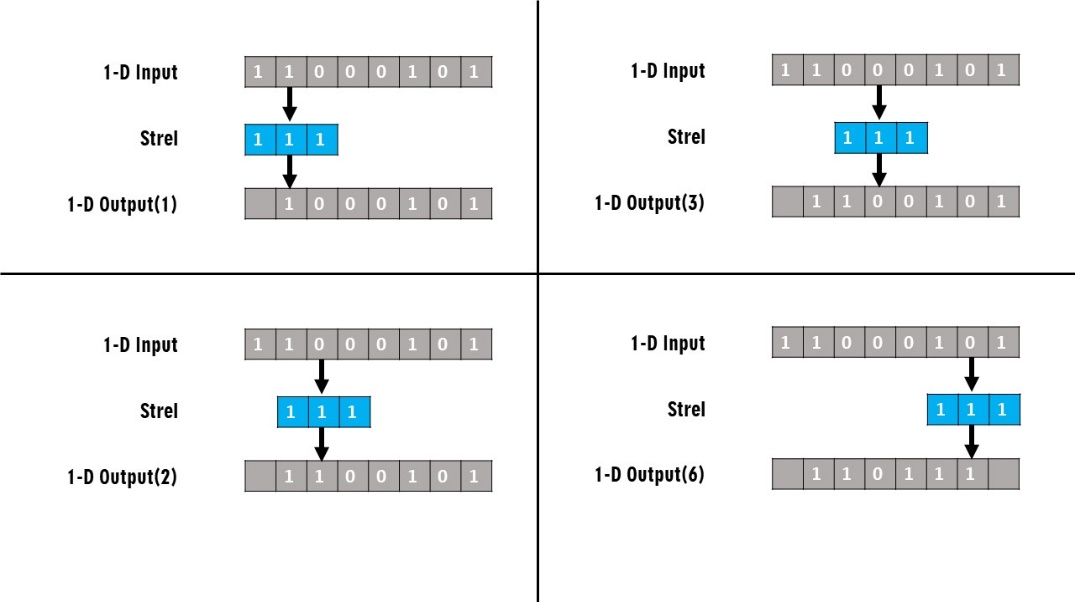
**Gambar 2.8.** Matriks Perubahan Matriks Sebelum dan Sesudah Penerapan Distance Transform (Fisher dkk, 2003)

1. Operasi Morfologi Citra

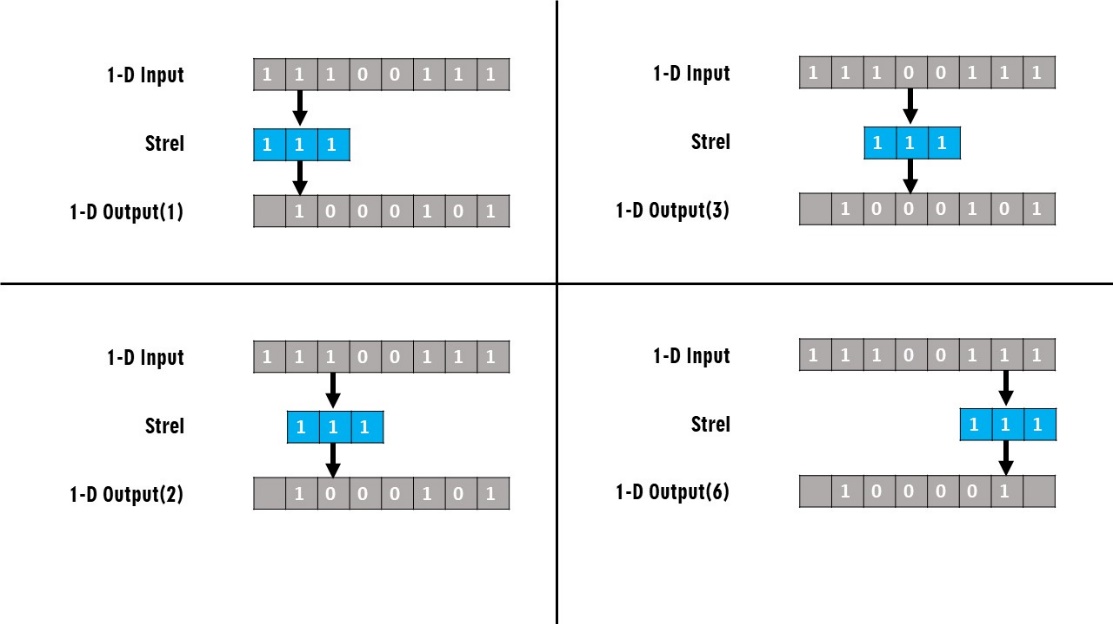
Operasi morfologi citra adalah teknik pengolahan citra yang didasarkan pada bentuk segmen atau region dalam citra. Operasi morfologi citra dilakukan dengan mengkonvolusi *structuring element* (strel) pada citra biner. Ukuran dan bentuk strel dapat ditentukan sesuai dengan citra keluaran yang diinginkan.

Terdapat beberapa jenis dari operasi morfologi citra seperti dilasi (*dilation*), erosi (*erosion*), pembukaan (*opening*), dan penutupan (*closing*). Dilasi bertujuan untuk memperbesar objek. Erosi bertujuan untuk mengecilkan objek. *Opening* merupakan operasi erosi yang diikuti oleh dilasi yang bertujuan untuk menghilangkan noise yang ada di luar objek. *Closing* merupakan operasi dilasi yang diikuti oleh erosi yang bertujuan untuk menghilangkan derau yang ada di dalam objek dengan menutup lubang yang ada di dalam objek.

Dalam menerapkan morfologi pada citra, terdapat dua operasi dasar yang digunakan, yaitu operasi *hit* dan operasi *fit.* Jika terdapat minimal satu dari nilai strel 1 yang cocok dengan masukan yang juga memiliki nilai *pixel* 1, maka akan diberikan keluaran 1. Selain itu, akan diberikan keluaran 0. Operasi ini merupakan operasi *hit* yang diilustrasikan pada **Gambar** **2.9**. Jika semua nilai strel 1 cocok dengan masukan yang juga memiliki nilai *pixel* 1, maka akan diberi keluaran 1. Selain itu, akan diberikan keluaran 0. Operasi ini merupakan operasi *fit* yang diilustrasikan pada **Gambar 2.10.**



**Gambar 2.9.** Contoh Operasi *Hit* pada Dilasi

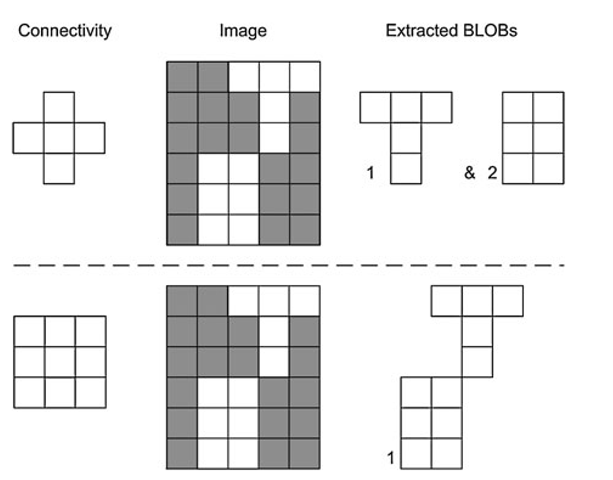


**Gambar 2.11.** Contoh Operasi *Fit* pada Erosi

1. *Binary Large Object* (Blob)

Secara umum, blob merupakan kumpulan data biner yang saling terhubung yang dikumpulkan dalam satu entitas. Blob dapat berupa gambar, audio, maupun objek multimedia lainnya. Dalam pengolahan citra digital, sebuah blob merujuk ke bagian dari citra yang memiliki properti yang konstan atau mendekati konstan sehingga bisa dianggap serupa satu sama lain. Blob pada citra terdiri atas kumpulan *pixel* yang saling terhubung. Penentuan kedua *pixel* terhubung atau tidak tergantung pada *connectivity* yaitu penentuan *pixel*-*pixel* mana yang bertetangga dan mana yang bukan.

Terdapat dua jenis *connectivity* yang paling sering digunakan yaitu *8-connectivity* dan *4-connectivity*. *8-connectivity* lebih akurat dibandingkan dengan *4-connectivity*. Tetapi, *4-connectivity* sering kali digunakan karena membutuhkan komputasi yang lebih sedikit dan memproses citra lebih cepat. Efek yang dihasilkan dari dua jenis *connectivity* ini diilustrasikan pada **Gambar 2.11**.



**Gambar 2.11.** Perbandingan Hasil Ekstraksi Blob dengan 4-*connectivity* dan 8-*connectivity*.

**2.3. Visi Komputer**

Visi komputer merupakan analisis otomatis terhadap gambar dan video yang dilakukan oleh komputer dalam rangka memperoleh pemahaman terhadap dunia. Visi komputer terinspirasi dari kemampuan sistem penglihatan manusia, ketika pertama kali diperkenalkan pada tahun 1960 dan 1970-an, visi komputer dianggap sebagai masalah yang cukup mudah dipecahkan.

Namun, hal itu dianggap mudah karena manusia memiliki sistem visualnya sendiri yang membuat tugas tersebut tampak mudah. Faktanya, sistem visual manusia sangat kompleks dan bahkan diperkirakan menggunakan fungsi otak manusia sebanyak 25% hingga lebih dari 50%. (Dawson-Howe, 2014)

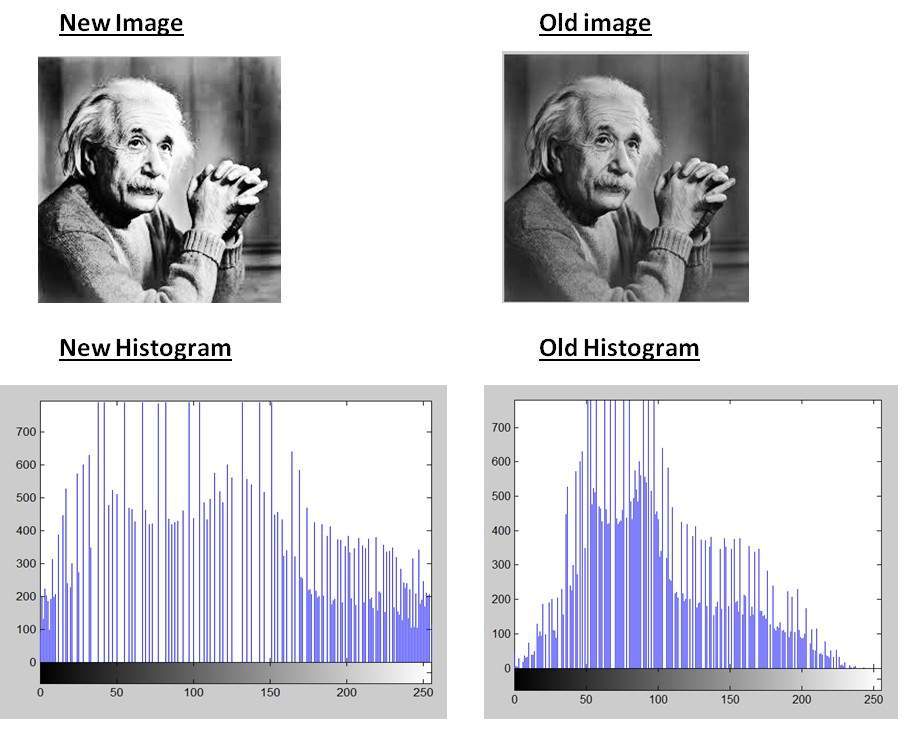
* 1. Histogram Citra

Histogram citra merupakan abstraksi dari citra di mana nilai frekuensi tiap citra (kecerahan/intensitas) ditentukan. Pada citra *grayscale* terdapat 256 intensitas skala keabuan yaitu bernilai 0 hingga 255. Nilai intensitas tersebut menunjukkan tingkat keabuan yang dimiliki masing-masing pixel dalam suatu citra. Histogram mengandung informasi mengenai citra dan informasi tersebut bersifat independen terhadap posisi dan orientasi objek dalam citra. Dalam beberapa kasus, histogram atau informasi yang diturunkan dari histogram (seperti rata-rata intensitas dan standar deviasi) bisa digunakan untuk melakukan klasifikasi. Namun, perlu diperhatikan bahwa histogram citra tidak unik dan memungkinkan banyak citra memiliki histogram yang serupa.

Citra terkadang sulit diamati oleh manusia dikarenakan keadaan citra yang terlalu gelap atau terlalu terang. Manusia dapat membedakan antara 700 hingga 900 tingkat keabuan dalam kondisi penglihatan optimal. Meskipun demikian, manusia bisa lebih mudah membedakan perbedaan intensitas yang lebih besar, jadi jika distribusi derajat keabuan dalam citra ditingkatkan, maka akan meningkatkan pemahaman observasi manusia.

Salah satu teknik untuk meningkatkan distribusi derajat keabuan dalam citra adalah dengan ekualisasi histogram (*histogram equalization*). Teknik ini berusaha mendistribusikan derajat keabuan dalam citra secara merata, sehingga menghasilkan histogram yang merata. Ekualisasi histogram bukan merupakan solusi yang baik (kecuali beberapa derajat keabuan yang umum pada citra masukan dipetakan ke beberapa skala keabuan berbeda pada citra keluaran), karena biasa menghasilkan histogram yang memiliki derajat keabuan yang tidak memiliki pasangan pixel, yang diselingi skala keabuan dengan nilai tinggi. (Dawson-Howe, 2014)

Contoh citra hasil ekualisasi histogram ditunjukkan pada **Gambar 2.12.** berikut.



**Gambar 2.12.** Ekualisasi histogram

* 1. *Smoothing*

Menghilangkan atau mengurangi derau pada gambar dapat dilakukan dengan berbagai metode. Metode paling umum yang digunakan untuk menghilangkan derau adalah *linear smoothing transformation* misalnya *Image Averaging* dan *Gaussian Smoothing*. Penekanan derau menggunakan *linear smoothing transformation* umumnya menghasilkan penghalusan pada tepi yang tajam. Selain transformasi linear, terdapat juga beberapa metode transformasi non-linear seperti Median Filter. (Dawson-Howe, 2014)

1. *Image Averaging*

Jika terdapat beberapa citra dengan kondisi yang sama persis, maka citra-citra tersebut dapat dirata-ratakan untuk mengurangi derau. Rata-rata citra dapat diekspresikan seperti persamaan (2.6) berikut:

|  |  |
| --- | --- |
|  | (2.6) |

1. *Local Averaging*

Jika hanya terdapat satu citra, perhitungan rata-rata tetap dapat dilakukan namun dalam lingkungan lokal. Bentuk paling sederhana dari teknik ini adalah menghitung rata-rata suatu titik dalam citra dengan 8 titik di sekitarnya, untuk setiap titik yang ada dalam citra.

*Local Averaging* merupakan proses perhitungan rata-rata lokal di mana setiap titik pada citra diberikan beban (*weight*) yang sama. Perubahan nilai beban pada saat perhitungan rata-rata juga dimungkinkan, hal ini sering juga disebut *Gaussian Distribution* dan perhitungan rata-rata menggunakan pembebanan ini disebut juga dengan *Gaussian Smoothing*.

Pengaplikasian filter seperti ini biasanya dilakukan dengan menggunakan teknik konvolusi, di mana *mask* konvolusi (h(i,j) merepresentasikan *weights* dari filter *smoothing*) akan dikonvolusi dengan citra masukan f(i,j) untuk menghasilkan citra keluaran f’(i,j). Konvolusi ini dinotasikan seperti persamaan (2.7) berikut:

|  |  |
| --- | --- |
|  | (2.7) |

1. *Median Filtering*

Median filter merupakan salah satu operasi smoothing non-linear, dimana operasi ini akan mengganti nilai tiap piksel dengan nilai median dari piksel-piksel dalam suatu wilayah kecil di sekitar piksel yang akan digantikan. Teknik ini bekerja cukup baik untuk menghilangkan noise, tanpa melakukan banyak blurring tepi dan dapat dilakukan secara berulang-ulang.

* 1. *Thresholding*

Citra *grayscale* umumnya memiliki 8 bit tiap pixel. Sementara pemrosesan citra *grayscale* lebih mudah dalam beberapa hal dibandingkan citra warna, terdapat bentuk citra yang lebih sederhana, yaitu citra biner, di mana prosesnya bahkan lebih mudah. Faktanya, sebagian besar dari aplikasi praktis dari visi komputer telah dikembangkan menggunakan citra biner.

Citra biner adalah citra yang hanya memiliki 1 bit untuk tiap pixelnya. Citra ini dibuat dengan menggunakan *threshold* di mana *threshold* yang digunakan ditentukan dalam berbagai cara. Citra biner yang dihasilkan biasanya diproses menggunakan morfologi matematika dan region biner yang disegmentasi diekstraksi dari citra menggunakan analisis komponen yang berkaitan.

Sebuah citra biner dibentuk dari citra *grayscale* dengan cara *thresholding*. Algoritma *thresholding* untuk citra biner adalah sebagai berikut persamaan (2.8):

|  |  |
| --- | --- |
|  | (2.8) |

Operasi *thresholding* umumnya digunakan untuk memisahkan *Object of Interest* dari background. *Object of Interest* umumnya direpresentasikan dengan nilai 1 (255) namun terkadang citra biner perlu diinvers untuk memperoleh *Object of Interest*. Contoh citra hasil *thresholding* dapat dilihat pada **Gambar 2.13** berikut.



**Gambar 2.13.** Citra *threshold*

**2.4. Jaringan Saraf Tiruan**

Jaringan saraf tiruan merupakan teknik pembelajaran mesin yang menyimulasikan mekanisme pembelajaran pada organisme biologis. Sistem saraf manusia mengandung sel, yang disebut sebagai neuron. Mekanisme biologis ini disimulasikan dalam jaringan saraf tiruan yang berisi unit perhitungan yang juga disebut neuron. Unit komputasi (neuron) tersebut terhubung satu sama lain melalui bobot, yang berperan seperti koneksi sinaptik dalam organisme biologis. Setiap masukan ke neuron diskalakan dengan bobot, yang mempengaruhi nilai fungsi yang dihitung pada unit tersebut.

Jaringan saraf tiruan menghitung fungsi masukan dengan menyebarkan nilai yang dihitung dari neuron masukan ke neuron keluaran dan menggunakan bobot sebagai parameter perantara. Pembelajaran terjadi dengan mengubah bobot yang menghubungkan neuron. Sama seperti rangsangan eksternal yang diperlukan untuk belajar dalam organisme biologis, rangsangan eksternal dalam jaringan saraf tiruan berupa data pelatihan yang berisi pasangan masukan dan keluaran dari fungsi yang harus dipelajari. Pasangan data pelatihan ini dimasukkan ke dalam jaringan saraf tiruan dengan menggunakan representasi masukan untuk membuat prediksi tentang label keluaran. Data pelatihan memberikan umpan balik terhadap nilai bobot dalam jaringan saraf tiruan berdasarkan pada seberapa baik keluaran yang dihasilkan untuk masukan tertentu. (Aggarwal, 2018)

Haykin menyatakan bahwa “Jaringan saraf adalah prosesor terdistribusi paralel besar-besaran yang memiliki kecenderungan alami untuk menyimpan pengetahuan pengalaman dan membuatnya tersedia untuk digunakan. Ini menyerupai otak dalam dua hal: 1. Pengetahuan diperoleh oleh jaringan melalui proses pembelajaran; 2. Kekuatan interkoneksi antar neuron, yang dikenal sebagai bobot sinaptik atau bobot, digunakan untuk menyimpan pengetahuan” (Samarasinghe, 2007) .

Berdasarkan cara belajarnya, jaringan saraf tiruan terbagi atas dua, yaitu *Supervised Learning* dan *Unsupervised Learning* (Duda, Hart, & Stork, 2000).

1. *Supervised Learning*

Pada *supervised learning*, seorang “guru” menyediakan kategori label atau *cost* untuk setiap pola dalam set pelatihan, dan mesin berusaha untuk mengurangi *cost* pada pola-pola tersebut (Graham et al., 2014).

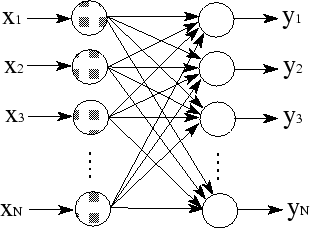
1. *Unsupervised Learning*

Pada *unsupervised learning* atau *clustering*, tidak ada “guru” spesifik, dan sistem akan membentuk *cluster* atau pengelompokan secara natural berdasarkan pola masukan. “Natural” selalu diartikan secara eksplisit maupun implisit dalam sistem pengelompokan itu sendiri, dan bila diberikan sebuah set pola tertentu, setiap algoritma *clustering* akan menghasilkan pengelompokan yang berbeda pula.

Sedangkan berdasarkan arsitekturnya, jaringan saraf tiruan terbagi atas dua, yaitu *Single Layer Network* dan *Multi Layer Network.*

1. *Single Layer Network*

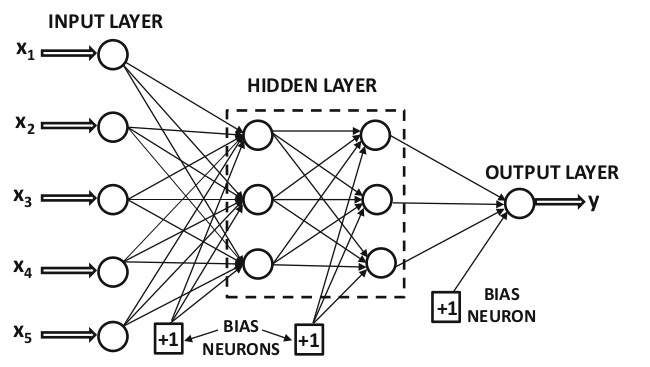
Arsitektur jaringan ini hanya terdiri dari satu *input layer* dan satu *output layer*. Setiap neuron yang terdapat di dalam *input layer* selalu terhubung dengan setiap neuron yang terdapat pada *output layer*. Jaringan ini hanya menerima masukan kemudian secara langsung akan mengolahnya menjadi suatu keluaran. **Gambar 2.14** menunjukkan arsitektur dari *single layer network*.



**Gambar 2.14.** *Single layer network*

1. *Multi Layer Network*

Arsitektur jaringan ini terdiri dari lebih dari satu lapisan komputasional. Berbeda dari *single layer network* yang hanya terdiri dari *input layer* dan keluaran, pada *multi layer network* terdapat lapisan komputasi tambahan di antara *input layer* dan *output layer*. Arsitektur dari jaringan saraf *multi layer* secara spesifik dinyatakan sebagai *feed forward network*, karena nilai pada neuron pada suatu lapisan berturut-turut dimasukkan ke neuron pada lapisan berikutnya, mulai dari *input layer* hingga pada *output layer*. **Gambar 2.15** menunjukkan arsitektur *multi layer network.*



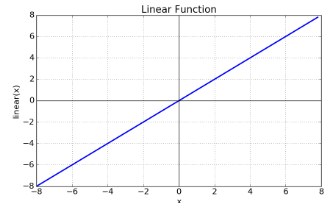
**Gambar 2.15.** *Multi-layer network*

**2.5. Activation Function**

*Activation function* merupakan sebuah node yang ditambahkan ke akhir *output* dari setiap jaringan syaraf. Hal ini juga dikenal sebagai *Transfer Function*. Hal ini juga dapat dilampirkan di antara dua *Neural Networks*. *Activation function* digunakan untuk menentukan *output* jaringan syaraf tiruan. *Activation function* memetakan nilai yang dihasilkan di antara 0 sampai 1 atau -1 sampai 1 dan lainlain (tergantung pada fungsinya). *Activation function* pada dasarnya dapat dibagi menjadi 2 tipe yaitu fungsi aktivasi linier dan fungsi aktivasi non linier. (Sharma,2017).

* + 1. **Fungsi Aktivasi Linear**

Seperti yang dapat dilihat pada **Gambar 2.16** fungsinya adalah garis atau linier. Oleh karena itu, *output* dari fungsi tidak akan dibatasi antara suatu rentang.

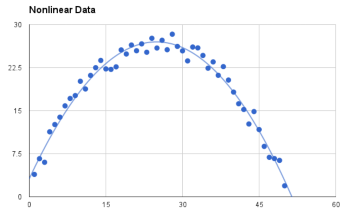


**Gambar 2.16.** Linear Function

Fungsi aktivasi pada *linear function* memiliki persamaan : f(x) = x dan rentang : -8 sampai 8. Fungsi aktivasi linier ini tidak membantu dengan kompleksitas atau berbagai parameter data biasa yang diumpankan ke jaringan syaraf tiruan. (Sharma, 2017).

* + 1. **Fungsi Aktivasi Nonlinear**

Fungsi Aktivasi Nonlinear adalah fungsi aktivasi yang paling banyak digunakan. Fungsi aktivasi non linier memudahkan model untuk mengeneralisasi atau menyesuaikan dengan berbagai data dan untuk membedakan antara *output*. Grafik nonlinier dapat dilihat seperti **Gambar 2.17** dibawah ini :

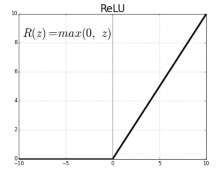


**Gambar 2.17.** *Non-linear Activation Function*

Terminologi utama yang perlu dipahami untuk fungsi nonlinier adalah Derivatif atau Diferensial dan Monotonik. Fungsi Derivatif atau Diferensial: Perubahan sumbu y w.r.t. Perubahan x-axis. Hal ini juga dikenal sebagai kemiringan dan fungsi Monotonik merupakan sebuah fungsi yang bervariasi sedemikian rupa sehingga tidak pernah berkurang atau tidak pernah meningkat. Terdapat beberapa fungsi aktivasi diantaranya adalah Sigmoid, Tanh, ReLU, dan Leaky ReLU.

* + 1. **ReLU (Rectrified Linear Unit)**

ReLU adalah fungsi aktivasi yang paling banyak digunakan di dunia saat ini. Karena, ini digunakan di hampir semua jaringan saraf konvolusi atau pembelajaran yang dalam. Fungsi ReLU dapat diilustrasikan seperti pada **Gambar 2.18** berikut.



**Gambar 2.18.** ReLU

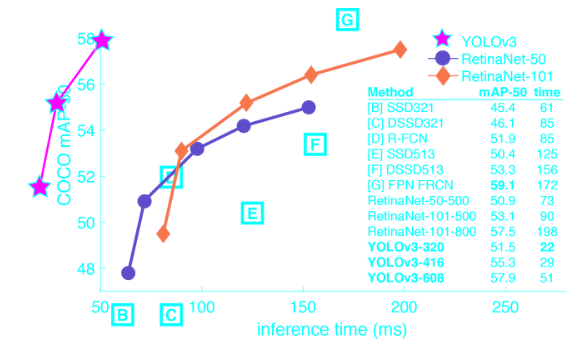
Seperti yang dapat dilihat, ReLU setengah diperbaiki (dari bawah). Ini adalah f (s) adalah nol bila z kurang dari nol dan f (z) sama dengan z bila z di atas atau sama dengan nol. Rentang: (0 sampai tak terbatas) Fungsi dan turunannya keduanya bersifat monoton. Tapi masalahnya adalah semua nilai negatif menjadi nol segera yang menurunkan kemampuan model agar sesuai atau melatih dari data dengan benar. Itu berarti setiap masukan negatif yang diberikan pada fungsi aktivasi ReLU mengubah nilainya menjadi nol segera dalam grafik, yang pada gilirannya mempengaruhi grafik yang dihasilkan dengan tidak memetakan nilai negatif secara tepat (Sharma, 2017).

**2.1.6. Algoritma You Only Look Once v3 (YOLO)**

Deteksi objek merupakan permasalahan yang jauh lebih kompleks dibandingkan proses klasifikasi, dimana sistem tidak hanya dapat mengenali objek, akan tetapi juga menunjukkan posisi di bagian mana objek tersebut berada, dan sistem tersebut tidak akan bekerja pada gambar dengan lebih dari satu objek.

*You Only Look Once,* atau biasa disingkat YOLO, adalah salah satu algoritma deep learning deteksi objek tercepat saat ini. Jika biasanya untuk melakukan proses deteksi objek, algoritma yang dibuat adalah dengan menggunakan algoritma klasifikasi seperti *VGGNet* atau *Inception* yang dijalankan pada setiap bagian kecil dari citra menggunakan *sliding window.*  Proses ini akan berhasil mendeteksi objek akan tetapi sistem akan sangat lambat, karena algoritma klasifikasi tersebut akan dijalankan berulang-ulang kali.

Algoritma YOLO melakukan pendekatan yang sangat berbeda dengan algoritma lain. Algoritma ini akan melakukan *scanning* pada citra hanya sekali dengan cara yang efisien. Untuk tingkat akurasinya, algoritma YOLO dapat bersaing dengan algoritma deteksi objek lain yang sudah ada. Berikut adalah grafik yang menunjukkan performa YOLO dibandingkan dengan algoritma lain. (Redmon et al., 2015)



**Gambar 2.19.** Grafik perbandingan performa YOLO

Dari grafik diatas, kita dapat melihat bahwa algoritma YOLO mendapatkan akurasi mean average precision (mAP) yang cukup tinggi, bersaing dengan algoritma lain, akan tetapi sangat unggul dalam *processing time* karena proses deteksinya yang menggunakan pendekatan yang sangat berbeda sehingga memangkas waktu komputasi.

* 1. **Penelitian Terkait**

Terdapat beberapa penelitian yang telah menerapkan teknik deep learning untuk mendeteksi objek-objek bawah laut. Juhwal Kim dan Son-Cheul Yu pada tahun 2016 membuat suatu sistem yang dapat mendeteksi keberadaan *Remote Operated Vehicle* (ROV) yang mengeksplorasi lautan dalam dengan menggunakan inputan data berupa *sonar-images*. Metode yang digunakan adalah *deep learning* (*convolutional neural network*) dengan menggunakan arsitektur YOLO (You Only Look Once). Hasilnya adalah sistem tersebut dapat memantau pergerakan ROV secara lebih efektif menggunakan kombinasi sonar-images dan arsitektur YOLO.

Kemudian untuk objek bulu babi dan timun laut, Danelle E. Cline dkk. melakukan penelitian dengan membuat suatu sistem pendeteksi dengan menggunakan iLab Neumorphic C++ Vision toolkit yang dikembangkan di University of California. Dengan menggunakan toolkit ini sistem yang dibangun mendapatkan akurasi 100% saat mendeteksi benthic echinoderm rathbunaster californicus (jenis bulu babi), dan 95% pada parasthicopus leukothele (jenis timun laut). Video bawah laut yang dianalisa merupakan video yang direkam menggunakan deep sea ROV.

Untuk mendeteksi dan mengklasifikasi jenis-jenis ikan, penelitian yang dilakukan oleh Lin Meng dkk. menggunakan metode *convolutional neural network* (CNN) untuk membedakan jenis-jenis ikan pada video yang direkam menggunakan underwater drone. Penelitian ini menghasilkan sebuah sistem pendeteksi dan klasifikasi jenis-jenis ikan dengan akurasi mencapai 87%. Berikut adalah **Tabel 2.1.** yang berisi penelitian terkait deteksi objek yang dilakukan di bawah laut.

**Tabel 2.1.** Penelitian Terkait

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Peneliti | Judul | Tahun | Metode | Hasil |
| Lin Meng, Takuya Hirayama, Shigeru Oyanagi | ***Underwater drone with panoramic camera for automatic fish recognition based on deep learning*** | 2018 | *Convolutional Neural Network* | Sistem yang mampu mendeteksi jenis ikan dengan akurasi mencapai 87%. |
| Peneliti | **Judul** | **Tahun** | **Metode** | **Hasil** |
| L.P.De Lima, F. C. Boogaard, R.E. De Graaf,dkk | ***Monitoring The Impacts Of Floating Structures On The Water Quality And Ecology Using An Underwater Drone*** | 2015 | Monitoring underwater dengan water quality sensor dan video camera | *Sistem monitoring menggunakan drone dengan kemampuan monitoring kondisi air yang tidak dapat dilihat secara langsung oleh mata manusia.* |
| Juhwan Kim and Son-Cheol Yu | ***Convolutional Neural Network-based Real-Time ROV Detection using Forward-looking Sonar Images*** | 2016 | YOLO | Sistem yang dapat memonitoring pergerakan ROV Vehicle pada saat mengeksplorasi laut yang dalam |
| Danelle E. Cline, Duane R. Edgington | **A Detection, Tracking, And Classification Sistem For Underwater Images** | 2015 | Klasifikasi menggunakan iLab Neuromorphic Vision C++ Toolkit | Sistem yang mendeteksi objek bulu babi dengan akurasi 100% dan timun laut(Parastichopus leukothele) dengan akurasi 95%. |
| Peneliti | **Judul** | **Tahun** | **Metode** | **Hasil** |
| Niranjan Anand Hule | ***Underwater Image Processing For Object Detection*** | 2016 | *LOG Filter* | Sistem yang dapat mendeteksi objek-objek dalam air dengan metode edge detection |

**BAB III  
METODOLOGI PENELITIAN**

* 1. **Tahapan Penelitian**

Sistem yang dibuat pada tugas akhir ini menggunakan algoritma *deep learning* YOLO generasi ketiga, yang terdiri atas *convolutional neural network layer* untuk proses ekstraksi fitur dari *input* serta proses *localization* objek, dan *fully connected layer* untuk mengklasifikasi jenis ubur-ubur. Adapun tahapan-tahapan yang akan dilakukan pada penelitian ini dapat dilihat pada **Gambar 3.1**.

**Gambar 3.1.** Diagram Tahapan Penelitian

Tahapan penelitian pada **Gambar 3.1.** dijelaskan sebagai berikut.

1. Studi Literatur merupakan tahapan awal dari penelitian ini. Tahap ini dilakukan untuk mengumpulkan penelitian terkait penggunaan teknologi deep learning untuk melakukan pendeteksian terhadap objek-objek yang berada di bawah laut.
2. Pengambilan data terbagi atas dua, untuk data training dan testing menggunakan data yang diambil dari dataset Open Images v5, sementara data video bawah laut untuk validasi diambil dari berbagai sumber data video dari Internet.
3. *Preprocessing data* berupa proses analisis data yang layak digunakan, melakukan penyortiran pada data, melakukan *labelling* terhadap data traning, untuk memperbaiki akurasi dari sistem yang dibuat.
4. Perancangan sistem dilakukan untuk mencari metode-metode deep learning yang akan digunakan dalam sistem. Dalam penelitian ini, metode yang digunakan untuk melakukan pendeteksian objek adalah YOLO generasi ketiga. Pada tahap ini juga dilakukan pembuatan *flowchart* terkait alur kerja sistem.
5. Implementasi Sistem dilakukan sesuai dengan *flowchart* yang telah dibuat pada tahap sebelmnya. Pada penelitian ini sistem dibuat dengan menggunakan bahasa pemrograman *python* dan *framework* Darknet untuk proses *training*.
6. Uji coba sistem dilakukan untuk mengetahui seberapa akurat sistem yang dibuat.
7. Tahap akhir dari penelitian ini adalah melakukan penulisan laporan penelitian dalam bentuk skripsi sebagai bahan publikasi.
   1. **Waktu dan Lokasi Penelitian**

Penelitian ini selama 9 bulan yang dimulai sejak disetujuinya proposal penelitian ini pada bulan Januari 2019 hingga proses pelaporan penelitian ini pada bulan September 2019. Penelitian ini dilakukan di Laboratorium Kecerdasan Buatan Departemen Teknik Informatika, Fakultas Teknik, Universitas Hasanuddin.

* 1. **Instrumen Penelitian**

Instrumen penelitian yang digunakan pada penelitian ini meliputi :

1. Software

a) Windows 10 x64

b) Jupyter Notebook Framework

c) Google Colab (Linux Ubuntu 18.04, VGA Tesla K4)

d) Python 3.6

1. Hardware
2. Laptop ASUS, RAM 8 GB, Prosesor Intel(R) Core(TM) i5-4210U CPU @2.20GHz
3. YI Action Cam @720p 60fps
4. TTRobotix *Seawolf Underwater Drone*
5. *Smartphone* Xiaomi A1
   1. **Teknik Pengambilan Data**

Data yang digunakan pada penelitian ini berupa citra dan video bawah laut dengan objek ubur-ubur. Data citra diperoleh dari mengekstraksi *dataset Open Images v4* yang memiliki *sub-class* data Ubur-ubur dengan jumlah data lebih dari 1000 citra, dari data tersebut akan dipilah kembali untuk mendapatkan citra dengan kualitas terbaik untuk digunakan.

* 1. **Perancangan Sistem**

Sistem ini secara umum dibagi menjadi dua bagian, yaitu pelatihan dan pengujian seperti yang diperlihatkan pada **Gambar 3.2**. Dalam proses pelatihan, model deep learning yang terdiri atas *convolutional neural netword* dan *fully connected layer*. Dalam proses pelatihan, fitur yang telah diekstrak dari data latih menggunakan *layer convolutional neural network* akan dijadikan input ke dalam *fully connected layer* yang akan menghasilkan hasil klasifikasi ubur-ubur. Hasil klasifikasi dari *fully connected layer* inilah yang akan digunakan untuk menentukan jenis ubur-ubur yang terdeteksi. Model yang sudah dilatih akan disimpan untuk kemudian digunakan untuk proses pengujian. Model deep learning yang sudah dilatih inilah yang akan digunakan untuk proses pengujian sistem nantinya.

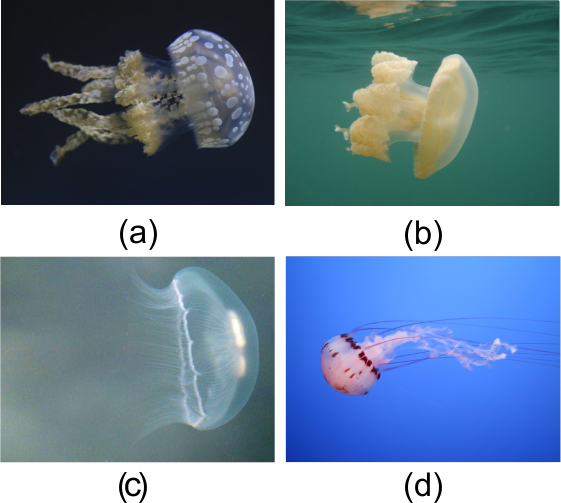


**Gambar 3.2.** Rancangan Sistem Deteksi Ubur-ubur

Berdasarkan **Gambar 3.2** diatas, rancangan sistem dapat diuraikan sebagai berikut.

**3.5.1. Input Citra**

Langkah pertama adalah mempersiapkan data citra ubur-ubur yang akan dijadikan output untuk proses pelatihan dan pengujian sistem. Dalam penelitian ini, data ubur-ubur yang digunakan diekstrak dari *dataset Open Images Dataset v4.* Data yang telah diekstrak kemudian dipisah menjadi 4 kelas ubur-ubur yaitu jenis Catostylus (*Blubber Jellyfish*)*,* Chrysaora fuscescens (*Sea Nettle*), *Mastigias Papua (Bell Jellyfish),* dan *Aurelia Aurita (Moon Jellyfish).*  **Gambar 3.3.** berikut adalah contoh jenis ubur-ubur yang dijadikan data latih.



**Gambar 3.3.** Contoh jenis ubur-ubur : (a) *Bell Jellyfish*, (b) *Blubber Jellyfish*, (c) *Moon Jellyfish*, dan (d) *Sea Nettle Jellyfish*

* + 1. **Preprocessing**

Metode *deep learning* YOLO v3 yang digunakan membutuhkan input citra dengan ukuran 416x416 piksel. Hal ini dikarenakan ukuran ini optimal untuk mendapatkan proses komputasi yang cepat tanpa harus kehilangan akurasi sistem karena kecilnya ukuran citra yang menjadi input. Oleh karena itu, dalam preprocessing sistem, baik citra maupun frame video inputan akan terlebih dahulu di*resize* ke dalam ukuran 416x416 piksel ini sebelum di*input* ke dalam *neural network* YOLO. Ilustrasi proses *resize*  citra dapat dilihat pada **Gambar 3.4.** berikut.



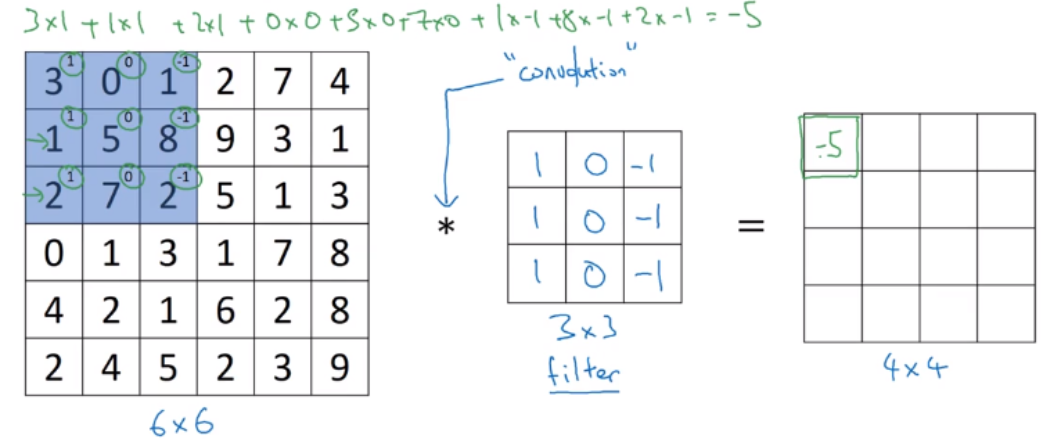
**Gambar 3.4.** Ilustrasi Proses *Resize* Citra

* + 1. **Feature Extraction**

Proses ekstraksi fitur dalam sistem ini menggunakan *Convolutional Neural Networks* (CNN). Dalam *convolutional Neural Network* ini, terdapat 2 proses yaitu *Convolution* dan *Max Pooling*.

* *Convolution*

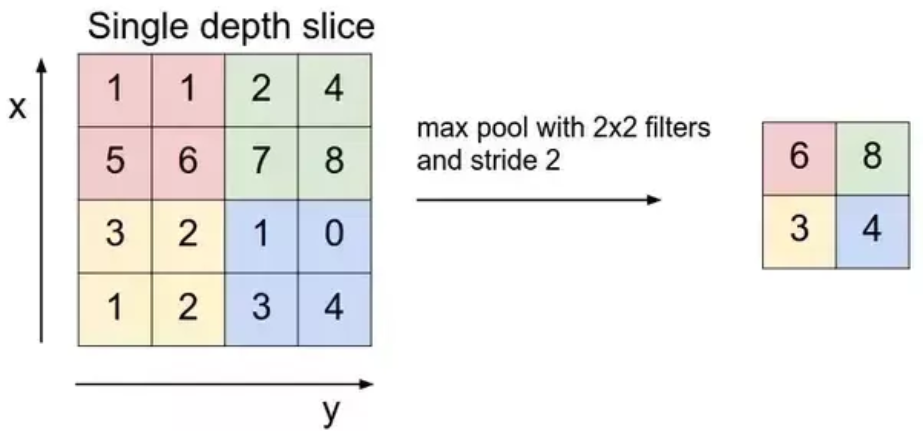
Layer pertama pada *feature extraction* adalah *convolutional layer*. *Convolutional layer* terdiri dari *neuron* yang membentuk sebuah filter dengan panjang dan tinggi (*pixels*). Filter ini akan digeser ke seluruh bagian gambar, setiap pergeseran akan dilakukan operasi dot antara input dan nilai dari filter tersebut sehingga menghasilkan sebuat output yang disebut activation map. Cara kerja lapisan ini dapat dilihat pada **Gambar 3.5.** berikut.



**Gambar 3.5.** *Convolutional Layer* (Brownlee, 2019)

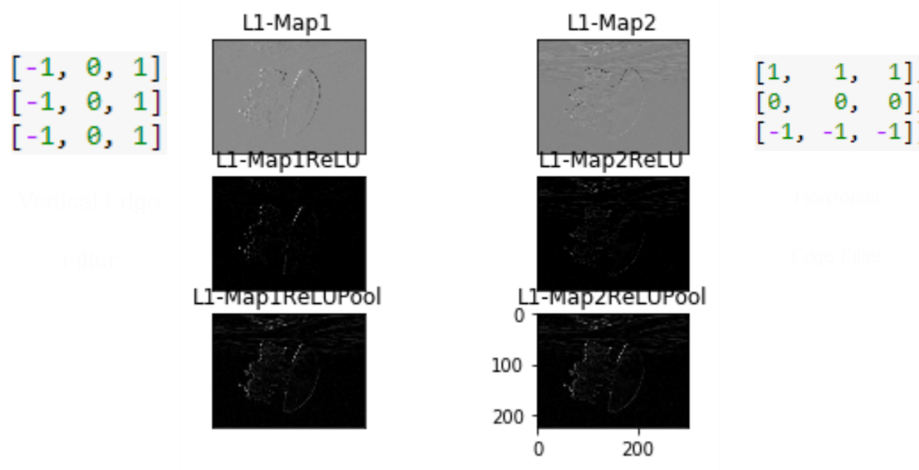
* *Max Pooling*

*Layer Max Pooling* terdiri dari sebuah filter dengan ukuran dan *stride* tertentu yang akan bergeser pada seluruh *activation map* yang berasal dari proses *convolution*. *Pooling layer* digunakan untuk mempercepat komputasi karena parameter yang harus diupdate akan menjadi semakin sedikit setelah proses pooling ini. Cara kerja proses ini dapat dilihat pada **Gambar 3.6.** berikut ini.

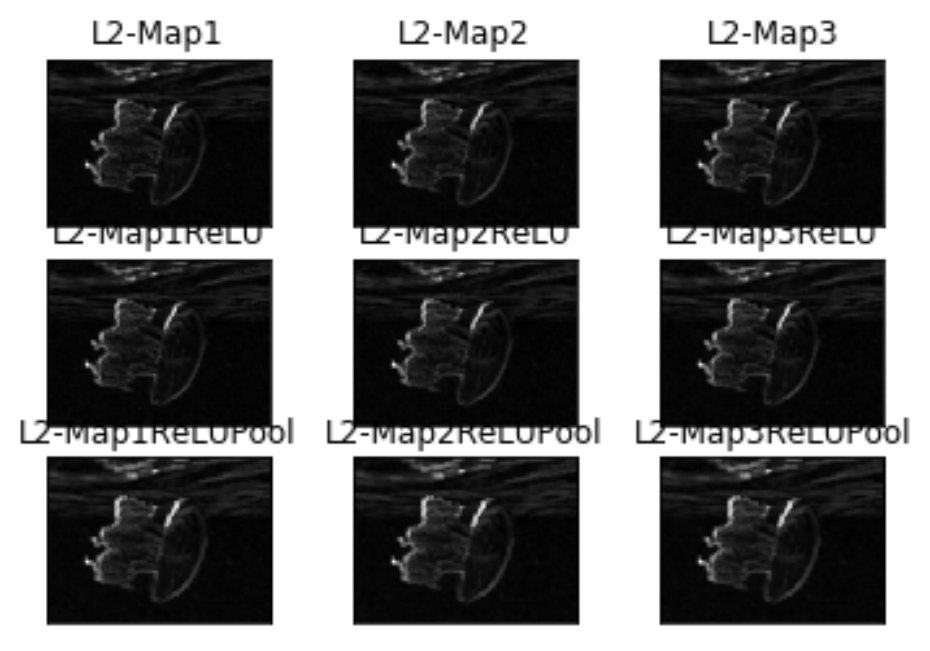


**Gambar 3.6.** *Max Pooling Layer* (Brownlee, 2019)

Dalam sistem yang dibuat, proses *convolution* dan *max pooling* akan dilakukan berulang-ulang hingga menjadi *activation map*. Activation Map merupakan citra dimana fitur-fitur objek sudah terlihat jelas dan objek dalam *activation map* tersebut telah siap untuk diklasifikasikan. Ilustrasi proses *convolution* dan *pooling*  dapat dilihat pada **Gambar 3.7** berikut. Pada gambar tersebut, citra inputan dioperasikan dengan filter berpola *vertical* dan *horizontal* yang mengekstrak fitur yang berbeda dari citra tersebut. **Gambar 3.8** menampilkan ilustrasi citra pada *convolutional* *layer* selanjutnya, dimana inputan dari layer tersebut berasal dari proses yang terjadi pada **Gambar 3.7**. Kemudian pada **Gambar 3.9.** menampilkan *activation map* hasil dari *convolutional layer* yang kemudian akan dijadikan input untuk *fully connected layer* untuk diklasifikasikan.



**Gambar 3.7.** Proses *convolution* dan *max pooling* pada citra.



**Gambar 3.8.** Proses *convolution* dan *max pooling* pada *convolutional layer* kedua.

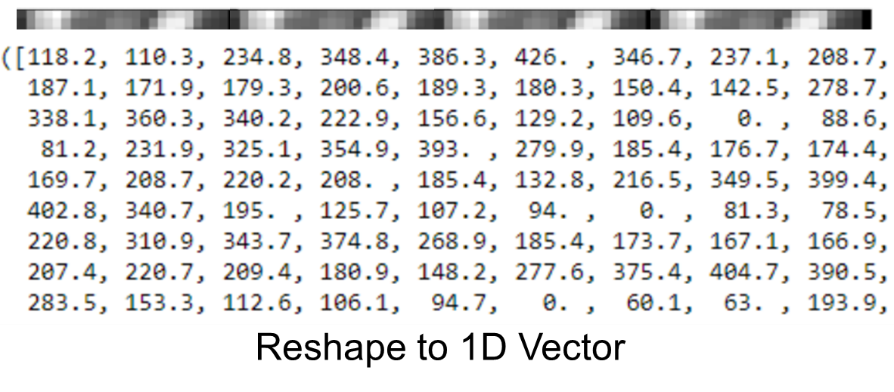


**Gambar 3.9.** *Activation map* dari proses *convolution* dan *max pooling*

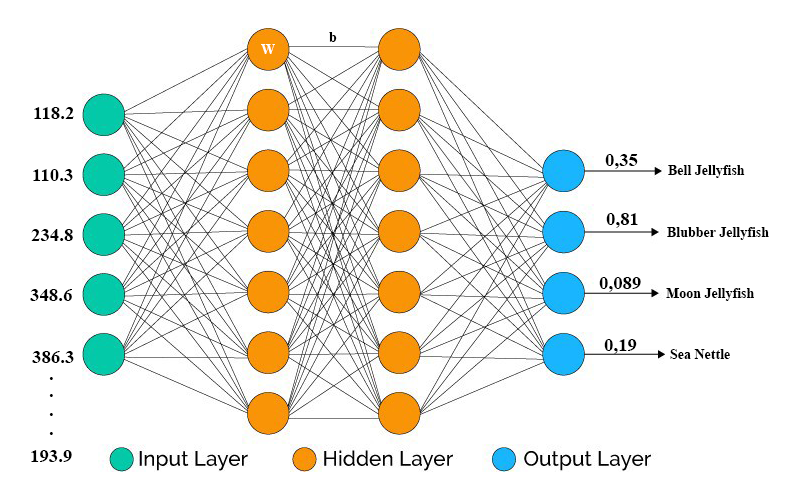
* + 1. ***Classification***

Data yang didapatkan dari ekstraksi fitur akan diteruskan ke proses klasifikasi. Dalam proses klasifikasi ini sistem menggunakan *neural network Fully Connected Layer* (*FC* *Layer*) atau biasa juga dikenal dengan nama *Multi Layer Perceptron* (MLP) seperti pada **Gambar 3.9** berikut. Jenis *neural network* ini memiliki *input layer*, *hidden layer*, *output layer*, *activation function*, dan *loss function*. Input dari *neural network* ini sendiri adalah bentuk vektor. *Activation map* yang dihasilkan dari *feature extraction layer* masih berbentuk *multidimensional array*, sehingga untuk bisa digunakan pada neural network ini, *activation map* tersebut perlu di*reshape* terlebih dahulu. **Gambar 3.10** berikut menunjukkan proses *reshape* *activation map* dari *multidimensional array* menjadi vektor untuk menjadi input pada FC *layer.* Setelah proses *reshape*, vektor akan dijadikan input untuk FC *layer* seperti pada **Gambar 3.11**.





**Gambar 3.10.** Proses *reshape activation map* menjadi dari *multidimensional array* menjadi vektor



**Gambar 3.11.** *Fully Connected Layer* pada Sistem

* + 1. **Output Prediction**

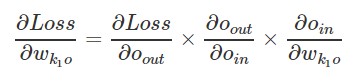
Output yang berasal dari *neural network* sebelumnya terdiri atas box dengan parameter seperti pada **Gambar 3.12** berikut. Selain itu ada juga nilai loss yang didapatkan dari proses fully connected layer, nilai ini kemudian digunakan untuk melakukan perbaruan pada nilai weight (nilai node pada *fully connected layer*) pada proses *backpropagation*.



**Gambar 3.12.** Output dari *Fully Connected Layer*

* + 1. **Backpropagation Weight Update**

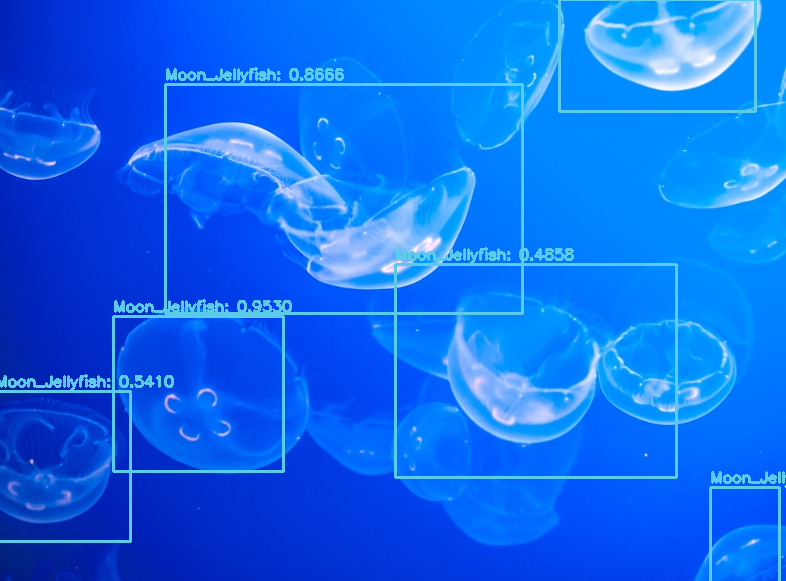
Proses *Backpropagation Weight Update* ini hanya terjadi saat proses training. Proses ini bermaksud untuk menyesuaikan kembali tiap parameter weight yang ada di dalam neural network. Pada tahapan ini menggunakan algoritma *Stochastic Gradient Descent (*SGD*),* algoritma ini cukup sederhana, yaitu dengan mengurangi nilai parameter dengan loss dan learning rate dari sistem yang dibuat. Untuk melakukan perubahan pada suatu nilai *weight* dalam *neural network*, digunakan *chain rule* dengan persamaan seperti di bawah ini.



(3,1)

* + 1. **Visualization**

Visualisasi hasil deteksi dari sistem akan ditampilkan dalam bentuk citra atau video secara real-time, proses visualisasi akan menggunakan opencv untuk mengefisienkan komputasi sehingga data lebih cepat diproses. **Gambar 3.13** berikut adalah merupakan contoh visualisasi output dari sistem yang ditampilkan menggunakan *module* OpenCV. Untuk visualisasi ini, sistem menggunakan parameter pada proses *output prediction* seperti pada **Gambar 3.12** di atas. Dalam sistem yang dibuat, nilai *objectness* score akan dijadikan acuan apakah bounding box akan digambar atau tidak. Pada sistem, nilai threshold untuk *objectness score* adalah 0.3. Jika nilai *objectness score* memenuhi, sistem akan menggambar bounding box menggunakan parameter *Box Co-ordinates,* kemudian menampilkan jenis kelas objek yang dideteksi beserta nilai *class score*nya. Kode program untuk unjuk kerja sistem dapat dilihat pada lampiran 1 dan lampiran 2.



**Gambar 3.13.** Visualisasi Output dari Sistem, *Blubber Jellyfish* (Kiri), *Moon Jellyfish* (Kanan).

* 1. **Analisis Kerja Sistem**

Analisis kerja sistem deteksi ubur-ubur pada video bawah laut dengan menggunakan metode deep learning dapat dilakukan dengan menghitung ketepatan prediksi yang dilakukan sistem terhadap data citra inputan pada proses testing dengan menggunakan bantuan *confussion matrix.*

Cara perhitungan nilai akurasi dapat dilihat pada **Gambar 3.14** berikut.



**Gambar 3.14.** Confusion matrix

*Confusion matrix* terdiri atas empat bagian, yaitu:

1. True Positif : yaitu jumlah prediksi benar dari data positif.
2. False Positif : yaitu jumlah prediksi salah dari data positif.
3. False Negatif : yaitu jumlah prediksi salah dari data negatif.
4. True Negatif : yaitu jumlah prediksi benar dari data negatif.

Untuk menghitung nilai akurasi sistem, dapat dilakukan dengan persamaan (3.3) berikut.

|  |  |
| --- | --- |
|  | (3.3) |

**BAB IV  
HASIL DAN PEMBAHASAN**

* 1. **Hasil Penelitian**

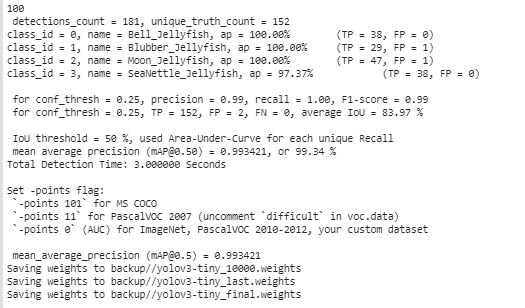
Pada bab ini, disajikan hasil kinerja sistem deteksi ubur-ubur pada video bawah laut dengan deep learning. Pada penelitian ini, confussion matrix digunakan untuk membantu dalam melakukan perhitungan akurasi sistem. Adapun jumlah data yang digunakan adalah sebanyak 100 citra, yang dibagi menjadi 80 data latih dan 20 data uji. Data pada saat training kemudian diduplikasi menjadi data baru dengan metode *data augmentation* agar data input menjadi lebih banyak sehingga dapat meningkatkan akurasi sistem. Proses pada *data augmentation* meliputi perubahan *angle* citra, perubahan nilai *exposure, hue,* hingga *saturation.*

Pada penelitian ini, dilakukan percobaan dengan menggunakan parameter batch, subdivision, momentum, width, height, channel, learning rate, dan iterations default. Nilai default untuk masing-masing parameter adalah batch 64, subdivision 16, momentum 0.09, width 416, height 416, channel 3, learning rate 0.001, dan iterations 10.000.

Adapun hasil kinerja sistem untuk percobaan menggunakan parameter-parameter diatas kemudian digambarkan ke dalam bentuk confusion matrix seperti pada **Tabel 4.1** berikut.

**Tabel 4.1.** Confusion Matriks pelatihan dengan 10.000 iterasi

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **ID** | **Nama** | **AP** | **TP** | **FP** | **FN** | **TN** |
| 0 | Bell\_Jellyfish | 100% | 38 | 0 | 0 | 0 |
| 1 | Blubber\_Jellyfish | 100% | 29 | 1 | 0 | 0 |
| 2 | Moon Jellyfish | 100% | 47 | 1 | 0 | 0 |
| 3 | Sea Nettle | 97% | 38 | 0 | 0 | 0 |

Sementara pada **Gambar 4.1.** berikut menunjukkan rincian dari proses training sistem dengan menggunakan 10.000 iterasi.

**Gambar 4.1.** Rincian hasil *training* sistem dengan 10.000 Iterasi

Berdasarkan **Tabel 4.1.** dan **Gambar 4.1.** yang telah dipaparkan diatas, dapat dilihat bahwa akurasi atau *mean average precision* yang diperoleh dari proses pelatihan sistem adalah sebesar 0.993421 atau 99,34%.

* 1. **Pembahasan**

Sistem yang dibuat menggunakan parameter-parameter berikut dalam proses pelatihan.

1. *Batch*

Parameter batch menunjukkan ukuran batch yang digunakan dalam proses training. Data training biasanya terdiri dari ratusan bahkan ribuan citra. Dalam sistem yang dibuat ini, nilai batch size adalah 64. Artinya, dalam satu iterasi terdapat 64 citra yang digunakan.untuk mengupdate parameter dari neural network.

1. *Subdivision*

Meskipun jumlah citra yang diproses dibatasi menjadi 64 citra dalam satu iterasi, hal ini belum cukup karena bisa saja GPU menjadi out of memory saat proses training. Dengan memberikan nilai subdivision menjadi 16, maka data yang akan diproses adalah 64/16 atau 4 citra dalam satu waktu. Hal ini dilakukan untuk menghindari error akibat GPU kekurangan memori. Saat testing, nilai batch dan subdivision akan di set menjadi 1.

1. *Width, Height, dan Channel*

Parameter-parameter ini menunjukkan ukuran citra inputan. Nilai untuk *width x height* yang digunakan adalah 416 x 416 piksel. Hasil deteksi bisa saja lebih baik jika ukuran input diubah menjadi 832 x 832 piksel misalnya, akan tetapi akan membutuhkan waktu training yang jauh lebih lama dan sistem pendeteksi akan menjadi lebih lambat. Sementara nilai *channel*=3 untuk menunjukkan bahwa citra yang akan diproses adalah citra 3-channel RGB (*Red, Green, Blu*e).

1. *Momentum*

Nilai weights yang terdapat pada sebuah neural network di update menggunakan batch citra tidak dengan keseluruhan data citra secara bersamaan. Hal ini akan menyebabkan fluktuasi pada proses update weight. Nilai momentum 0.9 diberikan sebagai pembatas bahwa perubahan nilai weights tidak boleh melebihi 0.9 dari weight sebelum proses update.

1. *Learning Rate, Steps, dan Scale*

Learning rate mengotrol seberapa agresif sistem akan belajar berdasarkan batch data inputannya. Biasanya nilai learning rate antara 0.01 hingga 0.0001. Pada sistem yang dibuat, nilai learning rate yang digunakan adalah 0.001.Di awal proses training learning rate diset tinggi kemudian semakin lama semakin kecil untuk mengurangi agresifitas proses learning. Steps adalah parameter yang digunakan sebagai threshold untuk learning rate, nilai steps pada sistem ini adalah 4000, sehingga nilai learning rate akan diperkecil dengan cara mengalikannya dengan parameter scale yang memiliki nilai 0.1 apabila iterasi sudah mencapai 4000 steps.

1. *Data Augmentation (angle, saturation, exposure, hue)*

Data input yang digunakan untuk pelatihan sistem ini adalah sebanyak 100 citra. Jumlah data tersebut sangat minim untuk proses pelatihan deep learning. Proses data augmentation memungkinkan untuk membuat data baru berdasarkan data yang sudah ada. Citra yang berjumlah 100 tersebut dapat digunakan untuk membuat lebih banyak data lagi untuk pelatihan, dengan cara mengubah sudut (*angle*) dari data yang ada, hingga mengubah saturasi, exposure, hingga nilai hue dari sebuah citra.

1. *Max Batches (Iterations)*

Parameter max batches menspesifikasikan seberapa banyak iterasi yang harus dilakukan untuk menyelesaikan proses training. Dalam sistem yang dibuat ini nilai max batches diset pada 10.000 iterasi.

Kode program untuk proses pelatihan dan konfigurasi parameter sistem secara lengkap dapat dilihat pada lampiran 3 dan lampiran 4. Dengan konfigurasi parameter-parameter diatas, sistem yang telah dibuat dilatih kemudian dilakukan proses uji yang memberikan hasil akurasi sebesar 99,34%.

**BAB V  
PENUTUP**

**5.1. Kesimpulan**

Dari hasil analisis yang telah dilakukan dalam pengujian sistem deteksi ubur-ubur pada video bawah laut dengan metode deep learning, dapat disimpulkan bahwa:

1. Sistem deteksi ubur-ubur pada video bawah laut dibuat dengan menggunakan algoritma deep learning YOLO v3 sebagai algoritma untuk mendeteksi objek. Metode deteksi ini menggunakan convolutional neural network untuk mengekstraksi fitur dari citra input. Fitur yang telah diekstraksi akan diteruskan ke multilayer perceptron untuk diklasifikasikan. Hasil klasifikasi dari multilayer perceptron ini selanjutnya akan divisualisasikan pada citra inputan berupa bounding box pada objek yang terdeteksi.
2. Hasil pengujian sistem pada iterasi 10.000, batch 64, subdivision 16, decay 0.0005, channel 3, input size 416x416, dan learning rate 0.001 menghasilkan nilai akurasi mencapai 99,34%

**5.2. Saran**

Sehubungan dengan selesainya proses pembuatan skripsi ini, penulis bermaksud menyampaikan beberapa saran kepada pembaca yakni:

1. Sistem yang telah dibuat ini dapat dikembangkan lebih lanjut terkait variasi jenis ubur-ubur yang dapat dideteksi oleh sistem dan meningkatkan akurasi dari sistem yang sudah ada. Sistem yang telah dibuat juga dapat dimodifikasi untuk mendeteksi objek lain selain ubur-ubur dengan menggantikan data latih untuk sistem untuk mengatasi permasalahan-permasalahan lainnya.
2. Terkait metode yang digunakan, pembaca dapat mencoba menggunakan berbagai metode deep learning untuk pendeteksian objek lainnya untuk menyelesaikan masalah yang sama, seperti metode Faster R-CNN, Faster R-CNN, hingga SSD yang memungkinkan untuk meningkatkan akurasi dari sistem yang dibuat.

# DAFTAR PUSTAKA

Graham, W.M., Gelcich, S., Robinson, K.L., Duarte, C.M., Brotz, L., Purcell, J.E., Madin, L.P., Mianzan, H., Sutherland, K.R., Uye, S., Pitt, K.A., Lucas, C.H., Bøgeberg, M., Brodeur, R.D., Condon, R.H., 2014. Linking human well-being and jellyfish: ecosistem services, impacts, and societal responses. Front. Ecol. Environ. 12, 515–523. https://doi.org/10.1890/130298

Hule, N.A., 2015. Underwater Image Processing For Object Detection. Int. J. Innov. Emerg. Res. Eng. 2, 5.

Ishii, H., 2003. Development time of planula larvae on the oral arms of the scyphomedusa Aurelia aurita. J. Plankton Res. 25, 1447–1450. https://doi.org/10.1093/plankt/fbg094

Kim, J., Yu, S.-C., 2016. Convolutional neural network-based real-time ROV detection using forward-looking sonar image, in: 2016 IEEE/OES Autonomous Underwater Vehicles (AUV). Presented at the 2016 IEEE/OES Autonomous Underwater Vehicles (AUV), IEEE, Tokyo, Japan, pp. 396–400. https://doi.org/10.1109/AUV.2016.7778702

L.P.De Lima, F. C. Boogaard, R.E. De Graaf,dkk, 2015. Monitoring The Impacts Of Floating Structures On The Water Quality And Ecology Using An Underwater Drone. E-Proc. 36 Th IAHR World Congr.

Lucas, C.H., Jones, D.O.B., Hollyhead, C.J., Condon, R.H., Duarte, C.M., Graham, W.M., Robinson, K.L., Pitt, K.A., Schildhauer, M., Regetz, J., 2014. Gelatinous zooplankton biomass in the global oceans: geographic variation and environmental drivers: Global gelatinous biomass. Glob. Ecol. Biogeogr. 23, 701–714. https://doi.org/10.1111/geb.12169

Meng, L., Hirayama, T., Oyanagi, S., 2018. Underwater-Drone With Panoramic Camera for Automatic Fish Recognition Based on Deep Learning. IEEE Access 6, 17880–17886. https://doi.org/10.1109/ACCESS.2018.2820326

Cline, D.E., Edgington, D.R., n.d. A DETECTION, TRACKING, AND CLASSIFICATION SISTEM FOR UNDERWATER IMAGES 4.

Alexey, 2019. Windows and Linux version of Darknet Yolo v3 & v2 Neural Networks for object detection (Tensor Cores are used): AlexeyAB/darknet.

Dawson-Howe, K. (2014). *A Practical Introduction to Computer Vision with OpenCV.* Wiley.

Samarasinghe, S. (2007). *Neural Networks for Applied Sciences and Engineering: From Fundamentals to Complex Pattern Recognition.* Taylor & Francis Group, LLC.

Redmon, J., Divvala, S., Girshick, R., Farhadi, A., 2015. You Only Look Once: Unified, Real-Time Object Detection. arXiv:1506.02640 [cs].

**LAMPIRAN**

**Lampiran 1** *Source Code* untuk unjuk kerja sistem pada video.

import numpy as np

import argparse

import cv2 as cv

import subprocess

import time

import os

from yolo\_utils import infer\_image, show\_image

FLAGS = []

if \_\_name\_\_ == '\_\_main\_\_':

parser = argparse.ArgumentParser()

parser.add\_argument('-m', '--model-path',

type=str,

default='./yolo3tiny-last/',

help='The directory where the model weights and configuration files are.')

parser.add\_argument('-w', '--weights',

type=str,

default='./yolo3tiny-last/yolo.weights',

help='Path to the file which contains the weights for YOLOv3.')

parser.add\_argument('-cfg', '--config',

type=str,

default='./yolo3tiny-last/yolo.cfg',

help='Path to the configuration file for the YOLOv3 model.')

parser.add\_argument('-i', '--image-path',

type=str,

help='The path to the image file')

parser.add\_argument('-v', '--video-path',

type=str,

help='The path to the video file')

parser.add\_argument('-vo', '--video-output-path',

type=str,

default='./output.avi',

help='The path of the output video file')

parser.add\_argument('-l', '--labels',

type=str,

default='./yolo3tiny-last/obj.names',

help='Path to the file having the labels in a new-line seperated way.')

parser.add\_argument('-c', '--confidence',

type=float,

default=0.2,

help='The model will reject boundaries which has a probabiity less than the confidence '

'value.default: 0.5')

parser.add\_argument('-th', '--threshold',

type=float,

default=0.3,

help='The threshold to use when applying the Non-Max Suppresion')

parser.add\_argument('-t', '--show-time',

type=bool,

default=False,

help='Show the time taken to infer each image.')

FLAGS, unparsed = parser.parse\_known\_args()

# Get the labels

labels = open(FLAGS.labels).read().strip().split('\n')

# Intializing colors to represent each label uniquely

colors = np.random.randint(0, 255, size=(len(labels), 3), dtype='uint8')

# Load the weights and configutation to form the pretrained YOLOv3 model

net = cv.dnn.readNetFromDarknet(FLAGS.config, FLAGS.weights)

# Get the output layer names of the model

layer\_names = net.getLayerNames()

layer\_names = [layer\_names[i[0] - 1] for i in net.getUnconnectedOutLayers()]

# If both image and video files are given then raise error

# Infer real-time on webcam

count = 0

#vid = cv.VideoCapture('videos/Danau ubur-ubur di kepulauan Togean. Indonesia.mp4')

vid = cv.VideoCapture('vid/Jellyfish\_3.mp4')

while True:

grabbed, frame = vid.read()

if not grabbed:

break

# preprociesing

# lab = cv.cvtColor(frame, cv.COLOR\_BGR2LAB)

# l, a, b = cv.split(lab)

# clahe = cv.createCLAHE(clipLimit=2.0, tileGridSize=(8, 8))

# cl = clahe.apply(l)

# limg = cv.merge((cl, a, b))

# frame = cv.cvtColor(limg, cv.COLOR\_LAB2BGR)

height, width = frame.shape[:2]

if count == 0:

frame, boxes, confidences, classids, idxs = infer\_image(net, layer\_names, height, width, frame, colors, labels, FLAGS)

count += 1

else:

frame, boxes, confidences, classids, idxs = infer\_image(net, layer\_names, height, width, frame, colors,

labels, FLAGS,

boxes, confidences, classids, idxs, infer=False)

count = (count + 1) % 6

cv.imshow('Jellyfish detection using Deep Learning', frame)

if cv.waitKey(1) & 0xFF == ord('q'):

break

vid.release()

cv.destroyAllWindows()

**Lampiran 2** *Source Code* untuk unjuk kerja sistem pada citra.

import numpy as np

import argparse

import time

import cv2

import os

ap = argparse.ArgumentParser()

ap.add\_argument("-i", "--image", required=True, help="path to input image")

ap.add\_argument("-y", "--yolo", required=True, help="base path to YOLO")

ap.add\_argument("-c", "--confidence", type=float, default=0.1, help="minimum score")

ap.add\_argument("-t", "--threshold", type=float, default=0.1, help="threshold of NMS")

args = vars(ap.parse\_args())

# load class label and YOLO models

labelsPath = os.path.sep.join([args["yolo"], "obj.names"])

LABELS = open(labelsPath).read().strip().split("\n")

# list color for represent layers

np.random.seed(42)

COLORS = np.random.randint(0, 255, size=(len(LABELS), 3), dtype="uint8")

weightsPath = os.path.sep.join([args["yolo"], "yolo.weights"])

configPath = os.path.sep.join([args["yolo"], "yolo.cfg"])

print("[INFO] loading yolo from disk...")

net = cv2.dnn.readNetFromDarknet(configPath, weightsPath)

# load images

image = cv2.imread(args["image"])

(H, W) = image.shape[:2]

# only output layer we need on YOLO

ln = net.getLayerNames()

ln = [ln[i[0] - 1] for i in net.getUnconnectedOutLayers()]

blob = cv2.dnn.blobFromImage(image, 1 / 255.0, (416, 416), swapRB=True, crop=False)

net.setInput(blob)

start = time.time()

layerOutputs = net.forward(ln)

end = time.time()

# show timing information on YOLO

print("[INFO] YOLO took {:.6f} seconds".format(end - start))

# initialize our lists of detected bounding boxes, confidences, and

# class IDs, respectively

boxes = []

confidences = []

classIDs = []

# loop over each of the layer outputs

for output in layerOutputs:

# loop over each of the detections

for detection in output:

# extract the class ID and confidence (i.e., probability) of

# the current object detection

scores = detection[5:]

classID = np.argmax(scores)

confidence = scores[classID]

# filter out weak predictions by ensuring the detected

# probability is greater than the minimum probability

if confidence > args["confidence"]:

# scale the bounding box coordinates back relative to the

# size of the image, keeping in mind that YOLO actually

# returns the center (x, y)-coordinates of the bounding

# box followed by the boxes' width and height

box = detection[0:4] \* np.array([W, H, W, H])

(centerX, centerY, width, height) = box.astype("int")

# use the center (x, y)-coordinates to derive the top and

# and left corner of the bounding box

x = int(centerX - (width / 2))

y = int(centerY - (height / 2))

# update our list of bounding box coordinates, confidences,

# and class IDs

boxes.append([x, y, int(width), int(height)])

confidences.append(float(confidence))

classIDs.append(classID)

# apply non-maxima suppression to suppress weak, overlapping bounding

# boxes

idxs = cv2.dnn.NMSBoxes(boxes, confidences, args["confidence"], args["threshold"])

# ensure at least one detection exists

if len(idxs) > 0:

# loop over the indexes we are keeping

for i in idxs.flatten():

# extract the bounding box coordinates

(x, y) = (boxes[i][0], boxes[i][1])

(w, h) = (boxes[i][2], boxes[i][3])

# draw a bounding box rectangle and label on the image

color = [int(c) for c in COLORS[classIDs[i]]]

cv2.rectangle(image, (x, y), (x + w, y + h), color, 2)

text = "{}: {:.4f}".format(LABELS[classIDs[i]], confidences[i])

cv2.putText(image, text, (x, y - 5), cv2.FONT\_HERSHEY\_SIMPLEX,

0.5, color, 2)

# show the output image

cv2.imshow("Image", image)

cv2.waitKey(0)

**Lampiran 3** *Source Code* Pelatihan YOLO v3 dengan Darknet

#!python3

"""

Python 3 wrapper for identifying objects in images

Requires DLL compilation

Both the GPU and no-GPU version should be compiled; the no-GPU version should be renamed "yolo\_cpp\_dll\_nogpu.dll".

On a GPU system, you can force CPU evaluation by any of:

- Set global variable DARKNET\_FORCE\_CPU to True

- Set environment variable CUDA\_VISIBLE\_DEVICES to -1

- Set environment variable "FORCE\_CPU" to "true"

To use, either run performDetect() after import, or modify the end of this file.

See the docstring of performDetect() for parameters.

Directly viewing or returning bounding-boxed images requires scikit-image to be installed (`pip install scikit-image`)

Original \*nix 2.7: https://github.com/pjreddie/darknet/blob/0f110834f4e18b30d5f101bf8f1724c34b7b83db/python/darknet.py

Windows Python 2.7 version: https://github.com/AlexeyAB/darknet/blob/fc496d52bf22a0bb257300d3c79be9cd80e722cb/build/darknet/x64/darknet.py

@author: Philip Kahn

@date: 20180503

"""

#pylint: disable=R, W0401, W0614, W0703

from ctypes import \*

import math

import random

import os

def sample(probs):

s = sum(probs)

probs = [a/s for a in probs]

r = random.uniform(0, 1)

for i in range(len(probs)):

r = r - probs[i]

if r <= 0:

return i

return len(probs)-1

def c\_array(ctype, values):

arr = (ctype\*len(values))()

arr[:] = values

return arr

class BOX(Structure):

\_fields\_ = [("x", c\_float),

("y", c\_float),

("w", c\_float),

("h", c\_float)]

class DETECTION(Structure):

\_fields\_ = [("bbox", BOX),

("classes", c\_int),

("prob", POINTER(c\_float)),

("mask", POINTER(c\_float)),

("objectness", c\_float),

("sort\_class", c\_int)]

class IMAGE(Structure):

\_fields\_ = [("w", c\_int),

("h", c\_int),

("c", c\_int),

("data", POINTER(c\_float))]

class METADATA(Structure):

\_fields\_ = [("classes", c\_int),

("names", POINTER(c\_char\_p))]

#lib = CDLL("/home/pjreddie/documents/darknet/libdarknet.so", RTLD\_GLOBAL)

#lib = CDLL("libdarknet.so", RTLD\_GLOBAL)

hasGPU = True

if os.name == "nt":

cwd = os.path.dirname(\_\_file\_\_)

os.environ['PATH'] = cwd + ';' + os.environ['PATH']

winGPUdll = os.path.join(cwd, "yolo\_cpp\_dll.dll")

winNoGPUdll = os.path.join(cwd, "yolo\_cpp\_dll\_nogpu.dll")

envKeys = list()

for k, v in os.environ.items():

envKeys.append(k)

try:

try:

tmp = os.environ["FORCE\_CPU"].lower()

if tmp in ["1", "true", "yes", "on"]:

raise ValueError("ForceCPU")

else:

print("Flag value '"+tmp+"' not forcing CPU mode")

except KeyError:

# We never set the flag

if 'CUDA\_VISIBLE\_DEVICES' in envKeys:

if int(os.environ['CUDA\_VISIBLE\_DEVICES']) < 0:

raise ValueError("ForceCPU")

try:

global DARKNET\_FORCE\_CPU

if DARKNET\_FORCE\_CPU:

raise ValueError("ForceCPU")

except NameError:

pass

# print(os.environ.keys())

# print("FORCE\_CPU flag undefined, proceeding with GPU")

if not os.path.exists(winGPUdll):

raise ValueError("NoDLL")

lib = CDLL(winGPUdll, RTLD\_GLOBAL)

except (KeyError, ValueError):

hasGPU = False

if os.path.exists(winNoGPUdll):

lib = CDLL(winNoGPUdll, RTLD\_GLOBAL)

print("Notice: CPU-only mode")

else:

# Try the other way, in case no\_gpu was

# compile but not renamed

lib = CDLL(winGPUdll, RTLD\_GLOBAL)

print("Environment variables indicated a CPU run, but we didn't find `"+winNoGPUdll+"`. Trying a GPU run anyway.")

else:

lib = CDLL("./libdarknet.so", RTLD\_GLOBAL)

lib.network\_width.argtypes = [c\_void\_p]

lib.network\_width.restype = c\_int

lib.network\_height.argtypes = [c\_void\_p]

lib.network\_height.restype = c\_int

copy\_image\_from\_bytes = lib.copy\_image\_from\_bytes

copy\_image\_from\_bytes.argtypes = [IMAGE,c\_char\_p]

def network\_width(net):

return lib.network\_width(net)

def network\_height(net):

return lib.network\_height(net)

predict = lib.network\_predict\_ptr

predict.argtypes = [c\_void\_p, POINTER(c\_float)]

predict.restype = POINTER(c\_float)

if hasGPU:

set\_gpu = lib.cuda\_set\_device

set\_gpu.argtypes = [c\_int]

make\_image = lib.make\_image

make\_image.argtypes = [c\_int, c\_int, c\_int]

make\_image.restype = IMAGE

get\_network\_boxes = lib.get\_network\_boxes

get\_network\_boxes.argtypes = [c\_void\_p, c\_int, c\_int, c\_float, c\_float, POINTER(c\_int), c\_int, POINTER(c\_int), c\_int]

get\_network\_boxes.restype = POINTER(DETECTION)

make\_network\_boxes = lib.make\_network\_boxes

make\_network\_boxes.argtypes = [c\_void\_p]

make\_network\_boxes.restype = POINTER(DETECTION)

free\_detections = lib.free\_detections

free\_detections.argtypes = [POINTER(DETECTION), c\_int]

free\_ptrs = lib.free\_ptrs

free\_ptrs.argtypes = [POINTER(c\_void\_p), c\_int]

network\_predict = lib.network\_predict\_ptr

network\_predict.argtypes = [c\_void\_p, POINTER(c\_float)]

reset\_rnn = lib.reset\_rnn

reset\_rnn.argtypes = [c\_void\_p]

load\_net = lib.load\_network

load\_net.argtypes = [c\_char\_p, c\_char\_p, c\_int]

load\_net.restype = c\_void\_p

load\_net\_custom = lib.load\_network\_custom

load\_net\_custom.argtypes = [c\_char\_p, c\_char\_p, c\_int, c\_int]

load\_net\_custom.restype = c\_void\_p

do\_nms\_obj = lib.do\_nms\_obj

do\_nms\_obj.argtypes = [POINTER(DETECTION), c\_int, c\_int, c\_float]

do\_nms\_sort = lib.do\_nms\_sort

do\_nms\_sort.argtypes = [POINTER(DETECTION), c\_int, c\_int, c\_float]

free\_image = lib.free\_image

free\_image.argtypes = [IMAGE]

letterbox\_image = lib.letterbox\_image

letterbox\_image.argtypes = [IMAGE, c\_int, c\_int]

letterbox\_image.restype = IMAGE

load\_meta = lib.get\_metadata

lib.get\_metadata.argtypes = [c\_char\_p]

lib.get\_metadata.restype = METADATA

load\_image = lib.load\_image\_color

load\_image.argtypes = [c\_char\_p, c\_int, c\_int]

load\_image.restype = IMAGE

rgbgr\_image = lib.rgbgr\_image

rgbgr\_image.argtypes = [IMAGE]

predict\_image = lib.network\_predict\_image

predict\_image.argtypes = [c\_void\_p, IMAGE]

predict\_image.restype = POINTER(c\_float)

predict\_image\_letterbox = lib.network\_predict\_image\_letterbox

predict\_image\_letterbox.argtypes = [c\_void\_p, IMAGE]

predict\_image\_letterbox.restype = POINTER(c\_float)

def array\_to\_image(arr):

import numpy as np

# need to return old values to avoid python freeing memory

arr = arr.transpose(2,0,1)

c = arr.shape[0]

h = arr.shape[1]

w = arr.shape[2]

arr = np.ascontiguousarray(arr.flat, dtype=np.float32) / 255.0

data = arr.ctypes.data\_as(POINTER(c\_float))

im = IMAGE(w,h,c,data)

return im, arr

def classify(net, meta, im):

out = predict\_image(net, im)

res = []

for i in range(meta.classes):

if altNames is None:

nameTag = meta.names[i]

else:

nameTag = altNames[i]

res.append((nameTag, out[i]))

res = sorted(res, key=lambda x: -x[1])

return res

def detect(net, meta, image, thresh=.5, hier\_thresh=.5, nms=.45, debug= False):

"""

Performs the meat of the detection

"""

#pylint: disable= C0321

im = load\_image(image, 0, 0)

if debug: print("Loaded image")

ret = detect\_image(net, meta, im, thresh, hier\_thresh, nms, debug)

free\_image(im)

if debug: print("freed image")

return ret

def detect\_image(net, meta, im, thresh=.5, hier\_thresh=.5, nms=.45, debug= False):

#import cv2

#custom\_image\_bgr = cv2.imread(image) # use: detect(,,imagePath,)

#custom\_image = cv2.cvtColor(custom\_image\_bgr, cv2.COLOR\_BGR2RGB)

#custom\_image = cv2.resize(custom\_image,(lib.network\_width(net), lib.network\_height(net)), interpolation = cv2.INTER\_LINEAR)

#import scipy.misc

#custom\_image = scipy.misc.imread(image)

#im, arr = array\_to\_image(custom\_image) # you should comment line below: free\_image(im)

num = c\_int(0)

if debug: print("Assigned num")

pnum = pointer(num)

if debug: print("Assigned pnum")

predict\_image(net, im)

letter\_box = 0

#predict\_image\_letterbox(net, im)

#letter\_box = 1

if debug: print("did prediction")

#dets = get\_network\_boxes(net, custom\_image\_bgr.shape[1], custom\_image\_bgr.shape[0], thresh, hier\_thresh, None, 0, pnum, letter\_box) # OpenCV

dets = get\_network\_boxes(net, im.w, im.h, thresh, hier\_thresh, None, 0, pnum, letter\_box)

if debug: print("Got dets")

num = pnum[0]

if debug: print("got zeroth index of pnum")

if nms:

do\_nms\_sort(dets, num, meta.classes, nms)

if debug: print("did sort")

res = []

if debug: print("about to range")

for j in range(num):

if debug: print("Ranging on "+str(j)+" of "+str(num))

if debug: print("Classes: "+str(meta), meta.classes, meta.names)

for i in range(meta.classes):

if debug: print("Class-ranging on "+str(i)+" of "+str(meta.classes)+"= "+str(dets[j].prob[i]))

if dets[j].prob[i] > 0:

b = dets[j].bbox

if altNames is None:

nameTag = meta.names[i]

else:

nameTag = altNames[i]

if debug:

print("Got bbox", b)

print(nameTag)

print(dets[j].prob[i])

print((b.x, b.y, b.w, b.h))

res.append((nameTag, dets[j].prob[i], (b.x, b.y, b.w, b.h)))

if debug: print("did range")

res = sorted(res, key=lambda x: -x[1])

if debug: print("did sort")

free\_detections(dets, num)

if debug: print("freed detections")

return res

netMain = None

metaMain = None

altNames = None

def performDetect(imagePath="data/dog.jpg", thresh= 0.25, configPath = "./cfg/yolov3.cfg", weightPath = "yolov3.weights", metaPath= "./cfg/coco.data", showImage= True, makeImageOnly = False, initOnly= False):

"""

Convenience function to handle the detection and returns of objects.

Displaying bounding boxes requires libraries scikit-image and numpy

Parameters

----------------

imagePath: str

Path to the image to evaluate. Raises ValueError if not found

thresh: float (default= 0.25)

The detection threshold

configPath: str

Path to the configuration file. Raises ValueError if not found

weightPath: str

Path to the weights file. Raises ValueError if not found

metaPath: str

Path to the data file. Raises ValueError if not found

showImage: bool (default= True)

Compute (and show) bounding boxes. Changes return.

makeImageOnly: bool (default= False)

If showImage is True, this won't actually \*show\* the image, but will create the array and return it.

initOnly: bool (default= False)

Only initialize globals. Don't actually run a prediction.

Returns

----------------------

When showImage is False, list of tuples like

('obj\_label', confidence, (bounding\_box\_x\_px, bounding\_box\_y\_px, bounding\_box\_width\_px, bounding\_box\_height\_px))

The X and Y coordinates are from the center of the bounding box. Subtract half the width or height to get the lower corner.

Otherwise, a dict with

{

"detections": as above

"image": a numpy array representing an image, compatible with scikit-image

"caption": an image caption

}

"""

# Import the global variables. This lets us instance Darknet once, then just call performDetect() again without instancing again

global metaMain, netMain, altNames #pylint: disable=W0603

assert 0 < thresh < 1, "Threshold should be a float between zero and one (non-inclusive)"

if not os.path.exists(configPath):

raise ValueError("Invalid config path `"+os.path.abspath(configPath)+"`")

if not os.path.exists(weightPath):

raise ValueError("Invalid weight path `"+os.path.abspath(weightPath)+"`")

if not os.path.exists(metaPath):

raise ValueError("Invalid data file path `"+os.path.abspath(metaPath)+"`")

if netMain is None:

netMain = load\_net\_custom(configPath.encode("ascii"), weightPath.encode("ascii"), 0, 1) # batch size = 1

if metaMain is None:

metaMain = load\_meta(metaPath.encode("ascii"))

if altNames is None:

# In Python 3, the metafile default access craps out on Windows (but not Linux)

# Read the names file and create a list to feed to detect

try:

with open(metaPath) as metaFH:

metaContents = metaFH.read()

import re

match = re.search("names \*= \*(.\*)$", metaContents, re.IGNORECASE | re.MULTILINE)

if match:

result = match.group(1)

else:

result = None

try:

if os.path.exists(result):

with open(result) as namesFH:

namesList = namesFH.read().strip().split("\n")

altNames = [x.strip() for x in namesList]

except TypeError:

pass

except Exception:

pass

if initOnly:

print("Initialized detector")

return None

if not os.path.exists(imagePath):

raise ValueError("Invalid image path `"+os.path.abspath(imagePath)+"`")

# Do the detection

#detections = detect(netMain, metaMain, imagePath, thresh) # if is used cv2.imread(image)

detections = detect(netMain, metaMain, imagePath.encode("ascii"), thresh)

if showImage:

try:

from skimage import io, draw

import numpy as np

image = io.imread(imagePath)

print("\*\*\* "+str(len(detections))+" Results, color coded by confidence \*\*\*")

imcaption = []

for detection in detections:

label = detection[0]

confidence = detection[1]

pstring = label+": "+str(np.rint(100 \* confidence))+"%"

imcaption.append(pstring)

print(pstring)

bounds = detection[2]

shape = image.shape

# x = shape[1]

# xExtent = int(x \* bounds[2] / 100)

# y = shape[0]

# yExtent = int(y \* bounds[3] / 100)

yExtent = int(bounds[3])

xEntent = int(bounds[2])

# Coordinates are around the center

xCoord = int(bounds[0] - bounds[2]/2)

yCoord = int(bounds[1] - bounds[3]/2)

boundingBox = [

[xCoord, yCoord],

[xCoord, yCoord + yExtent],

[xCoord + xEntent, yCoord + yExtent],

[xCoord + xEntent, yCoord]

]

# Wiggle it around to make a 3px border

rr, cc = draw.polygon\_perimeter([x[1] for x in boundingBox], [x[0] for x in boundingBox], shape= shape)

rr2, cc2 = draw.polygon\_perimeter([x[1] + 1 for x in boundingBox], [x[0] for x in boundingBox], shape= shape)

rr3, cc3 = draw.polygon\_perimeter([x[1] - 1 for x in boundingBox], [x[0] for x in boundingBox], shape= shape)

rr4, cc4 = draw.polygon\_perimeter([x[1] for x in boundingBox], [x[0] + 1 for x in boundingBox], shape= shape)

rr5, cc5 = draw.polygon\_perimeter([x[1] for x in boundingBox], [x[0] - 1 for x in boundingBox], shape= shape)

boxColor = (int(255 \* (1 - (confidence \*\* 2))), int(255 \* (confidence \*\* 2)), 0)

draw.set\_color(image, (rr, cc), boxColor, alpha= 0.8)

draw.set\_color(image, (rr2, cc2), boxColor, alpha= 0.8)

draw.set\_color(image, (rr3, cc3), boxColor, alpha= 0.8)

draw.set\_color(image, (rr4, cc4), boxColor, alpha= 0.8)

draw.set\_color(image, (rr5, cc5), boxColor, alpha= 0.8)

if not makeImageOnly:

io.imshow(image)

io.show()

detections = {

"detections": detections,

"image": image,

"caption": "\n<br/>".join(imcaption)

}

except Exception as e:

print("Unable to show image: "+str(e))

return detections

if \_\_name\_\_ == "\_\_main\_\_":

print(performDetect())

**Lampiran 4** *Source Code* Konfigurasi Pelatihan YOLO v3 dengan Darknet

[net]

# Testing

batch=64

subdivisions=16

# Training

# batch=64

# subdivisions=2

width=416

height=416

channels=3

momentum=0.9

decay=0.0005

angle=0

saturation = 1.5

exposure = 1.5

hue=.1

learning\_rate=0.001

burn\_in=1000

max\_batches = 10000

policy=steps

steps=2500,7500

scales=.1,.1

[convolutional]

batch\_normalize=1

filters=16

size=3

stride=1

pad=1

activation=leaky

[maxpool]

size=2

stride=2

[convolutional]

batch\_normalize=1

filters=32

size=3

stride=1

pad=1

activation=leaky

[maxpool]

size=2

stride=2

[convolutional]

batch\_normalize=1

filters=64

size=3

stride=1

pad=1

activation=leaky

[maxpool]

size=2

stride=2

[convolutional]

batch\_normalize=1

filters=128

size=3

stride=1

pad=1

activation=leaky

[maxpool]

size=2

stride=2

[convolutional]

batch\_normalize=1

filters=256

size=3

stride=1

pad=1

activation=leaky

[maxpool]

size=2

stride=2

[convolutional]

batch\_normalize=1

filters=512

size=3

stride=1

pad=1

activation=leaky

[maxpool]

size=2

stride=1

[convolutional]

batch\_normalize=1

filters=1024

size=3

stride=1

pad=1

activation=leaky

###########

[convolutional]

batch\_normalize=1

filters=256

size=1

stride=1

pad=1

activation=leaky

[convolutional]

batch\_normalize=1

filters=512

size=3

stride=1

pad=1

activation=leaky

[convolutional]

size=1

stride=1

pad=1

filters=27

activation=linear

[yolo]

mask = 3,4,5

anchors= 0.87,0.94, 1.36,1.31, 1.97,2.15, 3.74,3.76, 5.91,4.90, 11.63,10.52

classes=4

num=6

jitter=.3

ignore\_thresh = .7

truth\_thresh = 1

random=1

[route]

layers = -4

[convolutional]

batch\_normalize=1

filters=128

size=1

stride=1

pad=1

activation=leaky

[upsample]

stride=2

[route]

layers = -1, 8

[convolutional]

batch\_normalize=1

filters=256

size=3

stride=1

pad=1

activation=leaky

[convolutional]

size=1

stride=1

pad=1

filters=27

activation=linear

[yolo]

mask = 0,1,2

anchors = 0.87,0.94, 1.36,1.31, 1.97,2.15, 3.74,3.76, 5.91,4.90, 11.63,10.52

classes=4

num=6

jitter=.3

ignore\_thresh = .7

truth\_thresh = 1

random=1