**DETEKSI DAN PERHITUNGAN JUMLAH LARVA KEPITING RAJUNGAN DENGAN METODE *OBJECT DETECTION***

****

**TUGAS AKHIR**

*Disusun dalam rangka memenuhi salah satu persyaratan*

*Untuk menyelesaikan program Strata-1 Departemen Teknik Informatika*

*Fakultas Teknik Universitas Hasanuddin*

*Makassar*

**Disusun Oleh:**

**MUH. ARIEF WICAKSONO**

**D421 15 302**

**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 DAN PERHITUNGAN JUMLAH LARVA KEPITING RAJUNGAN DENGAN METODE *OBJECT DETECTION* ” 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 Andy Riswanto, S.H. dan Ibu Ir. Zaenab Muslimin, M.T. yang selalu menjadi motivasi penulis dari kecil hingga berada pada tahap ini dan menyelesaikan penyusunan tugas akhir ini;
2. Bapak Dr.Eng. Muhammad Niswar,S.T., M.I.T selaku pembimbing I yang senantiasa memberikan saran-saran serta bantuan selama proses pengambilan data hingga selesainya sistem ini dibuat, dan Ibu Dr.Eng Intan Sari Areni, S.T., M.T. selaku pembimbing II yang selalu menyediakan waktu untuk mengingatkan penulis untuk selalu semangat menyelesaikan penulisan penyusunan tugas akhir ini;
3. Bapak Dr. Amil Ahmad Ilham, ST., M.IT., selaku Ketua Departemen Teknik Informatika Fakultas Teknik Universitas Hasanuddin yang telah membimbing penulis selama masa perkuliahan;
4. Bapak Ir. Christoforus Yohannes, M.T., Ibu Dr. Ir. Ingrid Nurtanio, M.T., Bapak Dr. Ir. Zahir Zainuddin, M.Sc., Bapak Dr. Indrabayu, S.T., M.T., M.Bus.Sys., Bapak Dr. Adnan, S.T., M.T., Ibu Novy Nur R.A Mokobombang, S.T., Ms.TM., Ibu Elly Warni, S.T., M.T., Bapak Dr. Eng. Zulkifli Tahir, S.T., M.Sc., Bapak A. Ais Prayogi Alimuddin, S.T., M.Eng. selaku tenaga pengajar Departemen Teknik Informatika Fakultas Teknik Universitas Hasanuddin yang telah memberikan bimbingan dan ilmunya kepada penulis selama masa perkuliahan;
5. Bapak Ady Wahyudi Paundu, S.T., M.T., selaku tenaga pengajar Departemen Teknik Informatika Fakultas Teknik Universitas Hasanuddin yang telah membantu penulis dalam kelancaran *Research Internship* ke Jepang;
6. Kak Wahyudi, Kak Novi, dan Kak Furqan selaku mahasiswa S2 yang senantiasa membantu penulis dalam pengambilan data serta memberikan arahan kepada penulis dalam penyelesaian sistem pada tugas akhir ini;
7. Saudara Ryan Rafli, Fuad Khairi Hamid, Sabtian Juliana, Said Syamil Amas, Fadel Rezky Ramadhan, Muh. Zulfachril, Saudari Laura Natalia, Umniyah Nur Aprilyah, Charina, Reka Regina, Jusmiati, Khusnul Khatima selaku sahabat-sahabat SAPULIDI yang senantiasa memberikan semangat serta memotivasi penulis untuk selalu ceria hingga menyelesaikan penyusunan tugas akhir ini;
8. Teman-teman IoT Army yang senantiasa membantu penulis dalam masa perkuliahan;
9. Teman-teman lab UBICON yang senantiasa menemani dan menghibur selama masa penyusunan tugas akhir ini;
10. Saudara/saudari HYPERV15OR atas dukungan dan semangat yang diberikan kepada penulis;
11. Segenap Staff dan Dosen Departemen Teknik Informatika Fakultas Teknik Universitas Hasanuddin yang telah membantu penulis;
12. Putri Dwiyanti Mardan atas dukungannya selama ini;
13. Orang-orang berpengaruh lainnya yang tanpa sadar telah memotivasi dan membantu penulis.

Wassalam

Makassar, Desember 2019

Penulis

# ABSTRAK

Kepiting Rajungan merupakan salah satu jenis kepiting yang banyak terdapat di perairan payau Indonesia dan menjadi salah satu komoditas andalan Indonesia. Namun rajungan yang ada di alam dari waktu ke waktu terus mengalami penurunan, karena selama ini perdagangan rajungan dilakukan dengan mengambil stok langsung dari alam dan bukan berasal dari hasil budidaya. Maka dari itu Balai Perikanan Budidaya Air Payau (BPBAP) Takalar berupaya melakukan pembenihan rajungan secara massal untuk kepentingan pengembangan usaha budidaya, namun BPBAP Takalar menemukan beberapa tantangan, salah satunya terkait perhitungan *survival* *rate* dari larva rajungan yang diperkirakan sekitar 50%, nilai *survival* *rate* ini dihitung hanya dengan menggunakan perkiraan dan metode *sampling*. *Survival* *rate* yang dimaksudkan disini ialah jumlah larva rajungan dari fase *zoea* yang selanjutnya menjadi *megalopa*. Maka dari itu diperlukan adanya suatu sistem yang dapat mendeteksi larva rajungan serta menghitung jumlahnya, agar *survival* *rate* dari larva rajungan dapat dihitung secara jelas. Dengan menggunakan salah satu metode pada *machine* *learning* yaitu *deep* *learning*, maka sistem yang dapat mendeteksi dan menghitung larva kepiting rajungan dapat dibuat. Algoritma yang digunakan ialah YOLO (*You Only Look Once*) versi ke 3, jumlah data yang digunakan sebanyak 375 gambar dan dibagi menjadi 300 data *training* 75 data *testing* . Akurasi yang didapatkan dengan melakukan 2000 iterasi ialah sebesar 86.16%.

**Kata kunci :** larva, kepiting rajungan, *object detection,* *deep leraning, convolutional neural network,* YOLOV3, *counting.*

# DAFTAR ISI

[KATA PENGANTAR i](#_Toc26906089)

[ABSTRAK iv](#_Toc26906090)

[DAFTAR ISI v](#_Toc26906091)

[DAFTAR GAMBAR viii](#_Toc26906092)

[DAFTAR TABEL x](#_Toc26906093)

[BAB I 1](#_Toc26906094)

[1.1. Latar Belakang 1](#_Toc26906095)

[1.2. Rumusan Masalah 2](#_Toc26906096)

[1.3. Tujuan Penelitian 3](#_Toc26906097)

[1.4. Manfaat Penelitian 3](#_Toc26906098)

[1.5. Batasan Masalah 3](#_Toc26906099)

[1.6. Metode Penulisan 4](#_Toc26906100)

[1.6.1. Metode Pengambilan Data 4](#_Toc26906101)

[1.6.2. Studi Literatur 4](#_Toc26906102)

[1.6.3. Diskusi dan Konsultasi 4](#_Toc26906103)

[1.7. Sistematika Penulisan 5](#_Toc26906104)

[BAB II 7](#_Toc26906105)

[2.1. Kepiting Rajungan 7](#_Toc26906106)

[2.2. Pengolahan Citra 8](#_Toc26906107)

[2.2. *Machine* *Learning* 10](#_Toc26906108)

[*2.2.1.* *Training, Development, Testing Set* 11](#_Toc26906109)

[*2.2.2.* *Supervised Learning* 13](#_Toc26906110)

[2.2.3. Regresi 17](#_Toc26906111)

[2.3. *Convolutional* *Neural* *Network* 17](#_Toc26906112)

[2.3.1. Pooling 18](#_Toc26906113)

[2.4. Algoritma YOLO V3 (You Only Look Once) 20](#_Toc26906114)

[2.5. 23](#_Toc26906115)

[2.6. Penelitian terkait 23](#_Toc26906116)

[BAB III 27](#_Toc26906117)

[3.1. Tahapan Penelitian 27](#_Toc26906118)

[3.2. Waktu dan Lokasi Penelitian 29](#_Toc26906119)

[3.2. Instrumen Penelitian 29](#_Toc26906120)

[3.3. Teknik Pengambilan Data 29](#_Toc26906121)

[3.4. Perancangan Sistem 30](#_Toc26906122)

[*3.4.1.* *Input Image* 31](#_Toc26906127)

[*3.4.2.* *Preprocess* 34](#_Toc26906128)

[*3.4.3.* *Feature Extraction* 35](#_Toc26906129)

[*3.4.4.* *Classification* 38](#_Toc26906130)

[*3.4.5.* *Visualization* 39](#_Toc26906131)

[*3.4.6.* *Counting* 40](#_Toc26906132)

[3.5. Analisis kinerja sistem 40](#_Toc26906133)

[BAB IV 42](#_Toc26906134)

[1.7. Hasil Penelitian 42](#_Toc26906135)

[1.8. Pembahasan 45](#_Toc26906136)

[*1.8.1.* *Batch* 45](#_Toc26906146)

[*1.8.2.* *Max Batches (Iteration)* 45](#_Toc26906147)

[*1.8.3.* *Subdivision* 46](#_Toc26906148)

[*1.8.4.* *Width, Height dan Channel* 46](#_Toc26906149)

[*1.8.5.* *Learning Rate, Steps,* dan *Scale* 46](#_Toc26906150)

[*1.8.6.* *Data Augmentation (angle, saturation, exposure, hue)* 47](#_Toc26906151)

[BAB V 48](#_Toc26906152)

[5.1. Kesimpulan 48](#_Toc26906153)

[5.2. Saran 48](#_Toc26906154)

[DAFTAR PUSTAKA 50](#_Toc26906155)

[LAMPIRAN 51](#_Toc26906156)

# DAFTAR GAMBAR

**Gambar 2.1.** Kepiting Rajungan (Dictio.id, 2018) 7

**Gambar 2.2.** Citra Lena dan Citra Kapal (Rinaldi, 2004) 10

**Gambar 2.3.** *Supervised Learning* (Jan Wira, 2019) 14

**Gambar 2.4.**  *Supervised Learning – mathematical explanation* (Jan Wira, 2019) 15

**Gambar 2.5.** *Supervised Learning – mathematical explanation 2* (Jan Wira, 2019) 15

**Gambar 2.6.** *Supervised Learning Framework* (Jan Wira, 2019) 15

**Gambar 2.7.**  Ilustrasi  *binary classification* (Jan Wira, 2019) 16

**Gambar 2.8.** Ilustrasi *multi-label* dan *multi-class classification* (Jan Wira, 2019) 17

**Gambar 2.9.** *Sliding Window* (Jan Wira, 2019) 18

**Gambar 2.10.** Contoh *pooling* (Jan Wira, 2019) 19

**Gambar 2.11.** *Convolution* dan *pooling* (Jan Wira, 2019) 20

**Gambar 2.12.**  *Convolutional Neural Network* (Jan Wira, 2019) 20

**Gambar 2.13.** Perbandingan kinerja deteksi pada *COCO* *Dataset* (pjreddie.com, 2018) 21

**Gambar 2.14.**  Ilustrasi proses deteksi YOLO (pjreddie.com, 2018) 22

**Gambar 2.15.** Ilustrasi lapisan konvolusi YOLO V3 (Aditya, 2018) 23

**Gambar 3.1** Diagram Tahapan Penelitian 27

**Gambar 3.2** *Flowchart*  perancangan sistem deteksi dan perhitungan larva rajungan 31

**Gambar 3.3** Contoh gambar masukan 32

**Gambar 3.4.** *Interface* dari program *YOLO* *Mark* 33

**Gambar 3.5.** Contoh file anotasi dari program *YOLO Mark* 34

**Gambar 3.6** Ilustrasi proses *resize* gambar 35

**Gambar 3.7** Perkalian matriks gambar dan matriks filter 36

**Gambar 3.8.** *Output Matrix* 35

**Gambar 3.9** Pergeseran *filter* dengan *stride*=2 36

**Gambar 3.10.** *Fully Connected Layer* 39

**Gambar 4.1 G**ambar larva untuk data training 39

**Gambar 4.2.** Perhitungan akurasi model 41

# DAFTAR TABEL

**Tabel 2.1.** Penelitian terkait 25

**Tabel 3.1** *Confussion Matrix* 38

**Tabel 4.1.** Evaluasi nilai mAP sistem menggunakan *confussion matrix* 43

**Tabel 4.2.** Hasil pengujian sistem 44

# BAB I

**PENDAHULUAN**

## Latar Belakang

Rajungan (*Portunus* *pelagicus*) merupakan salah satu sumberdaya perikanan Indonesia yang mempunyai potensi besar untuk menjadi komoditas ekspor unggulan non migas. Permintaan rajungan terus mengalami peningkatan setiap tahunnya (Katisya dkk, 2017). Berdasarkan data dari Direktorat Jenderal Penguatan Daya Saing Produk Kelautan dan Perikanan terkait Kinerja Ekspor Produk Perikanan Indonesia Tahun 2018, pada periode Januari – September 2018, ekspor produk rajungan mencapai USD 370.14 juta atau sebesar 10.50% dari total nilai ekspor perikanan Indonesia, sedangkan dari sisi volume ekspor rajungan mencapai 21.57 ribu ton atau setara dengan 2.69% dari total volume ekspor perikanan Indonesia. Pasar utama produk Rajungan Indonesia adalah USA, Jepang, China, Malaysia dan Singapura. Berdasarkan data BPS, pasar USA menyerap 49.44% produk rajungan Indonesia dan menyumbang devisa sebesar USD 280.82 juta. Sebagian besar kebutuhan ekspor rajungan diperoleh dari hasil tangkapan di alam.

Namun tidak dapat dipungkiri bahwa populasi rajungan yang ada di alam dari waktu ke waktu terus mengalami penurunan sehingga upaya untuk menjaga populasi rajungan di alam adalah dengan melakukan kegiatan budidaya. Kegiatan pembenihan rajungan saat ini masih mengalami kendala karena hingga saat ini keberhasilan pembenihan rajungan menunjukkan tingkat *survival rate* (SR) dari larva rajungan masih rendah yaitu 25–30%. Saat ini teknologi pembenihan rajungan dikembangkan di 2 (dua) balai budidaya yaitu BPBAP (Balai Perikanan Budidaya Air Payau) Takalar Jepara dan BPBAP Takalar yang telah berhasil melakukan pembenihan rajungan secara massal, dan saat ini telah berhasil dikembangkan di masyarakat Kabupaten Demak, Jepara, Tarakan, Balikpapan, Belitung, Pangkalan Susu, dan Bangka.

Perhitungan *survival rate* dari larva rajungan dilakukan dengan menggunakan rumus dari Effendi (1979) yaitu Jumlah larva pada akhir pembesaran / Jumlah larva pada awal pembesaran 100X. Namun untuk menghitung jumlah larva pada awal dan akhir pembesaran, pembudidaya hanya menggunakan perhitungan manual menggunakan metode sampling serta melakukan perkiraan untuk menentukan jumlah dari larva rajungan. Hal ini akan membutuhkan waktu yang cukup lama serta ketelitian yang sangat tinggi, akan tetapi hasilnya tidak akurat.

Oleh karena itu pada tugas akhir ini dibangun sebuah sistem yang dapat mendeteksi dan menghitung larva kepiting rajungan dengan metode *Object* *Detection*. Untuk melakukan perhitungan, larva di kumpulkan dalam wadah 1 liter dan diambil gambarnya oleh kamera, hal ini di lakukan berulang kali hingga terkumpul 200 gambar dimana ada 200 liter air dalam kolam pembesaran. Sistem akan mendeteksi dan menghitung jumlah larva pada tiap gambar lalu menjumlahkan keseluruhan larva kepiting.

## Rumusan Masalah

Rumusan masalah yang akan diuraikan dalam penelitian ini antara lain :

1. Bagaimana cara mendeteksi dan menghitung larva rajungan dengan kamera di atas air menggunakan metode *object* *detection* ?
2. Bagaimana kinerja sistem deteksi dan perhitungan jumlah larva rajungan dalam suatu kolam pemeliharaan?

## Tujuan Penelitian

Tujuan dari penelitian ini adalah :

1. Untuk membuat sistem yang dapat digunakan mendeteksi dan menghitung larva rajungan menggunakan metode *object detection*
2. Untuk mengetahui kinerja sistem dalam mendeteksi dan menghitung larva rajungan dalam suatu kolam

## Manfaat Penelitian

Dengan dilakukannya penelitian ini, diharapkan manfaat yang didapatkan antara lain :

1. Bagi masyarakat yang dalam hal ini peternak atau pembudidaya kepiting rajungan, penelitian ini dapat digunakan sebagai sistem yang dapat digunakan untuk menghitung jumlah rajungan dengan efisien.
2. Bagi peneliti, penelitian ini dapat digunakan untuk menambah pengetahuan khsusnya dibidang kecerdasan buatan untuk mendeteksi objek terkhusus larva rajungan.
3. Bagi instansi pendidikan, penelitian ini dapat digunakan sebagai acuan dan referensi ilmiah untuk melakukan penelitian-penelitian lanjutan.

## Batasan Masalah

Adapun Batasan masalah dalam penelitian ini adalah :

1. Objek penelitian berupa larva rajungan di BPBAP Takalar.
2. Perhitungan jumlah larva kepiting rajungan dengan mendeteksi jumlah larva rajungan pada sebuah wadah untuk mewakili satu kolam.
3. Objek yang akan dideteksi berupa gambar.
4. Pengambilan data dilakukan pada siang hari dengan kondisi objek dapat terlihat dengan jelas.

## Metode Penulisan

Terdapat beberapa metode penulisan yang digunakan pada penelitian ini yaitu :

### Metode Pengambilan Data

Metode pengambilan data dilakukan dengan cara mengambil gambar objek secara langsung dengan mendatangi Balai Perikanan Budidaya Air Payau Takalar. Pengambilan gambar dilakukan dengan mengambil gambar larva rajungan pada sebuah wadah menggunakan kamera.

### Studi Literatur

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

### Diskusi dan Konsultasi

Diskusi dilakukan dengan melakukan diskusi bersama mahasiswa S2 yang sedang melakukan penelitian di tempat yang sama serta konsultasi secara langsung kepada dosen pembimbing serta pihak-pihak professional lainnya yang berhubungan dengan penelitian ini.

## Sistematika Penulisan

Laporan penelitian ini dibagi menjadi lima bab yang tersusun secara sistematis sebagai berikut:

1. **PENDAHULUAN**

Bab ini menjelaskan tentang latar belakang diangkatnya judul penelitian “Deteksi dan Perhitungan Jumlah Larva Kepiting Rajungan dengan Metode Object Detection”, disertai dengan rumusan masalah, tujuan penelitian, manfaat penelitian, batasan masalah, metode penulisan, dan sistematika penulisan.

1. **TINJAUAN PUSTAKA**

Bab ini membahas tentang teori-teori umum yang berkaitan dengan konsep dasar metode deteksi objek berbasis sistem jaringan saraf dan perhitungan jumlah objek yang terdeteksi.

1. **METODOLOGI PENELITIAN**

Bab ini memberikan gambaran mengenai perancangan sistem deteksi dan perhitungan jumlah larva kepiting rajungan dengan metode object detection beserta konsep perancangannya.

1. **HASIL DAN PEMBAHASAN**

Bab ini berisi tentang hasil deteksi dan hitung jumlah larva kepiting rajungan serta pembahasan yang dilengkapi dengan tabel hasil implementasi dan ujicoba sistem.

1. **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**

## Kepiting Rajungan

Rajungan adalah nama sekelompok kepiting dari beberapa marga anggota suku *Portunidae*. Jenis-jenis kepiting ini dapat berenang dan sepenuhnya hidup di laut. Rajungan sangat bergantung akan air. Dia tidak akan bisa tahan hidup tanpa air laut, dalam jangka waktu tertentu. Karena ketergantungannya itulah, si rajungan juga dijuluki sebagai si swimmer crab. Walaupun sih, sebenarnya dia lebih banyak ditemukan di berjalan di dasar laut, dibandingkan melayang di dalam air.

Secara umum morfologi kepiting rajungan berbeda dengan kepiting bakau (*Scylla* *serrata*), di mana si rajungan (*Portunus* *pelagicus*) memiliki bentuk tubuh yang lebih ramping dengan capit yang lebih panjang dan memiliki berbagai warna yang menarik pada karapasnya. Crajungan dapat dpilihat pada **Gambar 2.1.**



**Gambar 2.1.** Kepiting Rajungan (Dictio.id, 2018)

Dengan melihat warna dari karapas dan jumlah duri pada karapasnya, maka dengan mudah dapat dibedakan dengan kepiting bakau. Rajungan memiliki karapas berbentuk bulat pipih, sebelah kiri-kanan mata terdapat duri sembilan buah, di mana duri yang terakhir berukuran lebih panjang.

Di beberapa spot diving di nusantara, terutama di daerah secret bay, Gilimanuk, Bali, ternyata rajungan dianggap hama. Ini karena rajungan memangsa kuda laut dan frog fish yang menjadi obyek wisata selam di secret bay ini. Secara rutin, orang-orang adat di sekitar secret bay, yang juga mengelola spot diving secret bay, mencari dan menangkapnya. Selain untuk mengurangi populasinya yang memang sangat banyak, sehingga membahayakan populasi hewan unik lainnya, rajungan dicari juga untuk dimakan.

Saat ini, rajungan sudah mulai dikalengkan dan diekspor ke luar negeri. Karena hasilnya cukup menggiurkan, rajungan terus dicari dan ditangkap habis-habisan. Padahal keberadaan rajungan cukup penting untuk keberlangsungan ekosistem di laut. Dan karena eksploitasi itu pula, pemerintah mulai membudidayakan si rajungan, agar penangkapan di alam semakin berkurang.

## Pengolahan Citra

Data atau informasi tidak hanya disajikan dalam bentuk teks, tetapi juga dapat berupa gambar, audio (bunyi, suara, musik), dan video. Keempat macam data atau informasi ini sering disebut multimedia. Era teknologi informasi saat ini tidak dapat dipisahkan dari multimedia. Situs website di Internet dibuat semenarik mungkin dengan menyertakan visualisasi berupa gambar atau video yang dapat diputar. Beberapa waktu lalu istilah SMS (*Short Message Service*) begitu populer bagi pengguna telepon genggam (handphone atau HP). Tetapi, saat ini orang tidak hanya dapat mengirim pesan dalam bentuk teks, tetapi juga dapat mengirim pesan berupa gambar maupun video, yang dikenal dengan layanan MMS (*Multimedia Message Service*). (Rinaldi, 2004)

Secara harfiah, citra (*image*) adalah gambar pada bidang dwimatra (dua dimensi). **Gambar 2.2** adalah citra seorang gadis model yang bernama Lena, dan gambar di sebelah kanannya adalah citra kapal di sebuah pelabuhan. Ditinjau dari sudut pandang matematis, citra merupakan fungsi menerus (*continue*) dari intensitas cahaya pada bidang dwimatra. Sumber cahaya menerangi objek, objek memantulkan kembali sebagian dari berkas cahaya tersebut. Pantulan cahaya ini ditangkap oleh oleh alat-alat optik, misalnya mata pada manusia, kamera, pemindai (*scanner*), dan sebagainya, sehingga bayangan objek yang disebut citra tersebut terekam.

Citra sebagai keluaran dari suatu sistem perekaman data dapat bersifat :

* 1. optik berupa foto,
  2. analog berupa sinyal video seperti gambar pada monitor televisi,
  3. digital yang dapat langsung disimpan pada suatu pita magnetik.

Citra yang dimaksudkan adalah “citra diam” (*still images*). Citra diam adalah citra tunggal yang tidak bergerak. **Gambar 2.2** adalah contoh sebuah citra diam.

1. Lena (b) Kapal

**Gambar 2.2.** Citra (a) Lena dan Citra (b.) Kapal (Rinaldi, 2004)

Citra bergerak (*moving images*) adalah rangkaian citra diam yang ditampilkan secara beruntun (sekuensial) sehingga memberi kesan pada mata 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 atas ratusan sampai ribuan *frame*.

## *Machine* *Learning*

*Machine* *learning*, yaitu teknik untuk melakukan inferensi terhadap data dengan pendekatan matematis. Inti *machine learning* adalah untuk membuat model (matematis) yang mereﬂeksikan pola-pola data. (Jan Wira, 2019)

*Machine* *learning* sama-sama melakukan inferensi, tetapi pada representasi yang berbeda. Inferensi pada bidang keilmuan representasi pengetahuan mencakup tentang bagaimana cara (langkah dan proses) mendapatkan sebuah keputusan, diberikan premis. pada *machine* *learning*, inferensi yang dimaksud lebih ke ranah hubungan variabel. misalnya, apakah penjualan akan meningkat apabila biaya marketing ditingkatkan. Bila kamu ingat dengan mata pelajaran matematika SMA (logika preposisi), kamu sadar bahwa membuat sistem cerdas menggunakan representasi pengetahuan simbolik itu susah. Representasi pengetahuan secara tradisional dianggap relatif kurang *scalable*, khususnya apabila bekerja dengan data yang besar. Sementara itu, *machine* *learning* berada pada daerah representasi data/ilmu/pengetahuan dalam bentuk matematis karena keilmuan *machine* *learning* diturunkan dari matematika dan statistika.

### *Training, Development, Testing Set*

Terdapat dua istilah penting dalam pembangunan model *machine* *learning* yaitu: *training* dan *testing* . *Training* adalah proses membangun model dan *testing* adalah proses menguji kinerja model pembelajaran. *Dataset* adalah kumpulan data (sampel dalam statistik). Sampel ini adalah data yang digunakan untuk membuat model maupun mengevaluasi model *machine* *learning*. Umumnya, *dataset* dibagi menjadi tiga jenis yang tidak beririsan (satu sampel pada himpunan tertentu tidak muncul pada himpunan lainnya):

1. *Training* *set* adalah himpunan data yang digunakan untuk melatih atau membangun model.
2. *Development* *set* atau *validation* *set* adalah himpunan data yang digunakan untuk mengoptimisasi saat melatih model. Model dilatih menggunakan *training* *set* dan pada umumnya kinerja saat latihan diuji dengan *development* *set*. Hal ini berguna untuk generalisasi (agar model mampu mengenali pola secara generik).
3. *Testing* *set* adalah himpunan data yang digunakan untuk menguji model setelah proses latihan selesai. *Testing* *set* adalah *unseen* *data*. Artinya, model dan manusia tidak boleh melihat sampel ini saat proses latihan. Banyak orang yang tergoda untuk melihat *testing* *set* saat proses latihan walaupun itu adalah tingkah laku yang buruk karena menyebabkan bias.

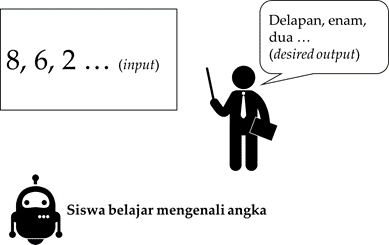
Satu sampel pada himpunan data disebut sebagai *data* *point* atau *instance* yang merepresentasikan suatu kejadian statistik (*event*). Perlu diingat, *training*, *development*, dan *testing* *data* diambil (*sampled* ) dari distribusi yang sama dan memiliki karakteristik yang sama (*independently* *and* *identically* *distributed* ). Distribusi pada masing-masing dataset ini juga sebaiknya seimbang (*balanced* ) dan memuat seluruh kasus. Misal, sebuah dataset *binary* *classification* sebaiknya memuat 50% kasus positif dan 50% kasus negatif.

Pada umumnya, rasio pembagian *dataset* adalah (80% : 10% : 10%) atau (90% : 5% : 5%) (*training* : *development* : *testing* ). Development set pada umumnya bisa tidak digunakan apabila *dataset* berukuran kecil (hanya dibagi menjadi *training* dan *testing* *set* saja). Dalam kasus ini, pembagian dataset menjadi *training* dan *testing* *set* pada umumnya memiliki rasio (90% : 10%), (80% : 20%), (70% : 30%), atau (50% : 50%). Pada kasus ini, kinerja saat *training* diuji menggunakan *training* *set* (dikenal sebagai *closed* *testing* ).

Saat tidak menggunakan *development* *set* (hanya ada *training* dan *testing* *set* ), opsi lain untuk mengevaluasi model dengan metode K *cross*-*validation*. Artinya, *training* *dataset* dibagi menjadi K bagian. K bagian digunakan untuk *training*, kemudian menguji kinerja model saat latihan (*validation*) menggunakan satu bagian. Hal ini diulangi sebanyak K kali dimana sebuah bagian data digunakan sebagai testing set sebanyak sekali (bergilir).

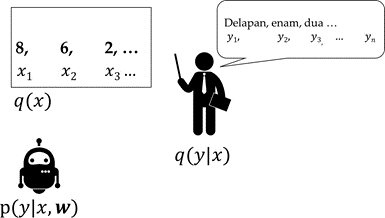
### *Supervised Learning*

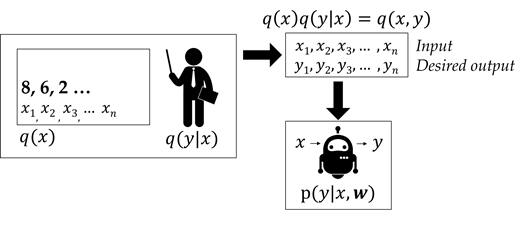
*supervised* *learning* adalah pembelajaran terarah/terawasi. Artinya, pada pembelajaran ini, ada guru yang mengajar (mengarahkan) dan siswa yang diajar. Manusia berperan sebagai guru, kemudian mesin berperan sebagai siswa. Perhatikan **Gambar 2.3** sebagai ilustrasi, seorang guru menuliskan angka di papan “8, 6, 2” sebagai contoh untuk siswanya, kemudian gurunya memberikan cara mem- baca yang benar untuk masing-masing angka. Contoh angka melambangkan *input*, kemudian cara membaca melambangkan *desired* *output*. Pasangan *input* *desired* *output* ini disebut sebagai *instance* (untuk kasus *supervised* *learning* ). Pembelajaran metode ini disebut *supervised* karena ada yang memberikan contoh jawaban (*desired* *output* ).



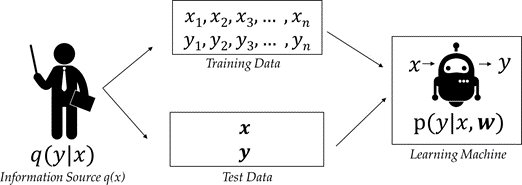
**Gambar 2.3.** *Supervised Learning* (Jan Wira, 2019)

Perhatikan **Gambar 2.4** dan **Gambar 2.5**, x adalah kejadian (*event* *random* *variable*), untuk *event* tertentu dapat dinotasikan sebagai . *x* dapat berupa vektor, teks, gambar, dan lain sebagainya (perhatikan konteks pembahasan buku). Seorang guru sudah mempunyai jawaban yang benar untuk masing-masing contoh dengan suatu fungsi distribusi probabilitas kondisional (*conditional* *probability* *density* *function*) baca: *function* *q* *for* *y* *given* x, melambangkan hasil yang benar/diharapkan untuk suatu *event*. Siswa (mesin) mempelajari tiap pasang pasangan *input*-*desired* *output* (*training* *data*) dengan mengoptimalkan *conditional* *probability* *density* *function* , dimana *y* adalah target (*output*), *x* adalah input dan vektor *w* adalah *learning* *parameters*. Proses belajar ini, yaitu mengoptimalkan *w* disebut sebagai *training*. Proses *training* bertujuan untuk mengaproksimasi *q(y | x)* melalui *p(y | x, w).*

  
**Gambar 2.4.**  *Supervised Learning – mathematical explanation* (Jan Wira, 2019)

**

**Gambar 2.5.** *Supervised Learning – mathematical explanation 2* (Jan Wira, 2019)

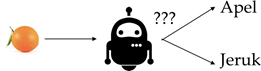
****

**Gambar 2.6.** *Supervised Learning Framework* (Jan Wira, 2019)

Pada **Gambar 2.6** model memiliki panah ke *training* *data* dan *test* *data*, artinya model hasil *training* sangat bergantung pada data dan guru. Model yang dihasilkan *training* (hasil pembelajaran kemampuan siswa) untuk data yang sama bisa berbeda untuk guru yang berbeda.

Tujuan *supervised* *learning*, secara umum untuk melakukan klasifikasi (classification). Misalkan mengklasifikasikan gambar buah (apa nama buah pada gambar), diilsutrasikan pada **Gambar 2.7**. Apabila hanya ada dua kategori, disebut *binary* *classification*. Sedangkan bila terdapat lebih dari dua kategori, disebut *multi-class* *classification*. Contoh *multi-class* *classification* adalah mengklasifikasikan gambar buah ke dalam himpunan kelas: apel, mangga atau sirsak.

Ada tipe klasifikasi lain disebut *multi-label* *classification* yaitu ketika ingin mengklasifikasikan suatu instans ke dalam suatu himpunan kelas. Perbedaan *multi-class* dan *multi-label* *classification* agak *tricky*. Pada *multi-class* *classification*, suatu instans hanya bisa berkorespondensi dengan satu kelas. Sedangkan pada *multi-label* *classification*, satu instans dapat berkorespondensi dengan lebih dari satu kelas. Misalnya, suatu berita dapat masuk ke kategori agama dan politik pada waktu bersamaan. Artinya, label pada *multi-class* *classification* bersifat *mutually* *exclusive*, sedangkan label tidak bersifat *mutually* *exclusive* pada *multi-label* *classification*. Perhatikan **Gambar 2.8** sebagai ilustrasi, dimana setiap baris merepresentasikan kelas yang berkorespondensi dengan setiap instans, nilai “1” melambangkan *TRUE* dan nilai “0” melambangkan *FALSE*. *Multi-label* *classification* dapat didekomposisi menjadi beberapa *Binary* *classification*, yaitu mengklasifikasikan apakah instans dapat di-*assign* ke suatu kelas atau tidak.



**Gambar 2.7.**  Ilustrasi  *binary classification* (Jan Wira, 2019)



**Gambar 2.8.** Ilustrasi *multi-label* dan *multi-class classification* (Jan Wira, 2019)

Pemahaman *supervised* *learning* adalah mengingat Ada tiga hal penting pada *supervised* *learning* yaitu *input*, *desired* *output*, dan *learning* *parameters*. Perlu ditekankan *learning* *parameters* berjumlah lebih dari satu, dan sering direpresentasikan dengan vektor (*bold* ) atau matriks. Berdasarkan model yang dibuat, dapat dilakukan klasifikasi (misal simbol yang ditulis di papan adalah angka berapa). Secara konseptual, klasifikasi didefinisikan sebagai persamaan 2.1 yaitu memilih label (kelas/kategori y) paling optimal dari sekumpulan label C, diberikan (*given*) suatu instans data tertentu.

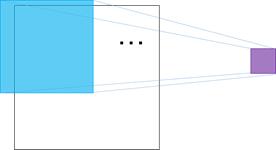
(2.1)

### Regresi

Pada persoalan regresi, prediksi output berupa bilangan kontinu. Misalnya pada regresi suatu fungsi polinomial, fungsi f(*x*) diberikan data *{(x1, y1), · · · , (xN , yN )}*. Setelah itu, digunakan fungsi aproksimasi untuk mencari tahu nilai *yN+1* dari data baru *xN+1*. Perbedaan regresi dan klasifikasi adalah pada tipe output. Untuk regresi, tipe output adalah nilai kontinu; sementara tipe output pada persoalan klasifikasi adalah suatu objek pada himpunan (i.e., memilih opsi pada himpunan jawaban).

## *Convolutional* *Neural* *Network*

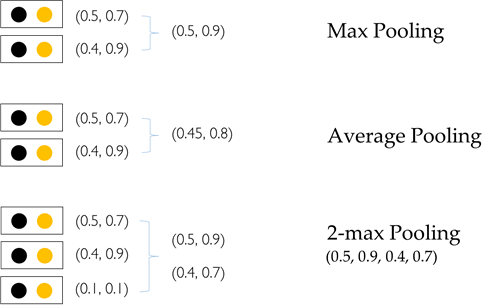
CNN (*Convolutional* *Neural* *Network*) adalah kemampuan mengenali aspek yang *informatif* pada regional tertentu (lokal). Dibanding meng*copy* mesin pembelajaran beberapa kali untuk mengenali objek pada banyak regional, ide lebih baik adalah untuk menggunakan *sliding* *window*, setiap operasi pada *window* 1 bertujuan untuk mencari aspek lokal yang paling informatif. Ilustrasi diberikan oleh **Gambar 2.9**. Warna biru merepresentasikan satu *window*, kemudian kotak ungu merepresentasikan aspek lokal paling informatif (disebut *filter* ) yang dikenali oleh *window*. Dengan kata lain, suatu *window* mentransformasi menjadi suatu nilai numerik (filter ). Suatu *window* (regional) dapat ditransformasi menjadi d nilai numerik (*d-channels*, setiap elemen berkorespondensi pada suatu *filter* ). *Window* ini kemudian digeser-geser sebanyak *T* kali, sehingga akhirnya didapatkan vektor dengan Panjang *d \* T .* Keseluruhan operasi ini disebut sebagai *convolution.*

**

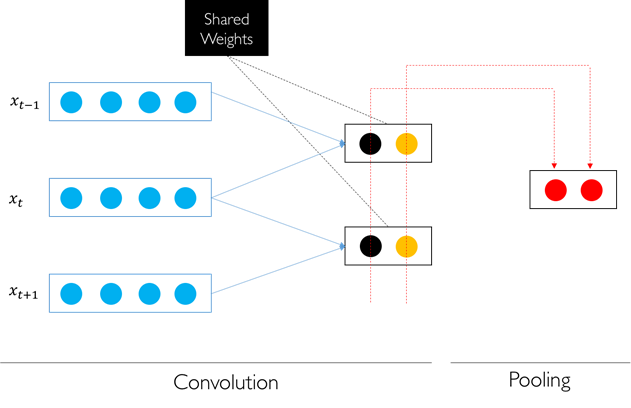
**Gambar 2.9.** *Sliding Window* (Jan Wira, 2019)

### Pooling

Pada tahap *convolution*, Setiap *k-sized* *window* diubah menjadi satu vektor berdimensi d (yang dapat disusun menjadi matriks D). Semua vek- tor yang dihasilkan pada tahap sebelumnya dikombinasikan (*pooled* ) menjadi satu vektor c. Ide utamanya adalah mengekstrak informasi paling informatif (semacam meringkas). Ada beberapa teknik *pooling*, diantaranya: *max* *pooling*, *average* *pooling*, dan *K-max* *pooling* 3; diilustrasikan pada **Gambar 2.10**. *Max* *pooling* mencari nilai maksimum untuk setiap dimensi vektor. *Average* *pooling* mencari nilai rata-rata tiap dimensi. *K-max* *pooling* mencari K nilai terbesar untuk setiap dimensinya (kemudian hasilnya digabungkan). Gabungan operasi *convolution* dan *pooling* secara konseptual diilustrasikan pada **Gambar 2.11**.

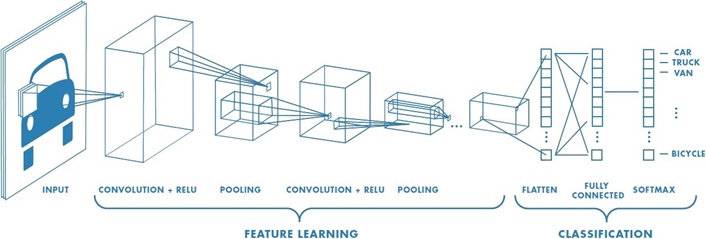


**Gambar 2.10.** Contoh *pooling* (Jan Wira, 2019)

**

**Gambar 2.11.** *Convolution* dan *pooling* (Jan Wira, 2019)

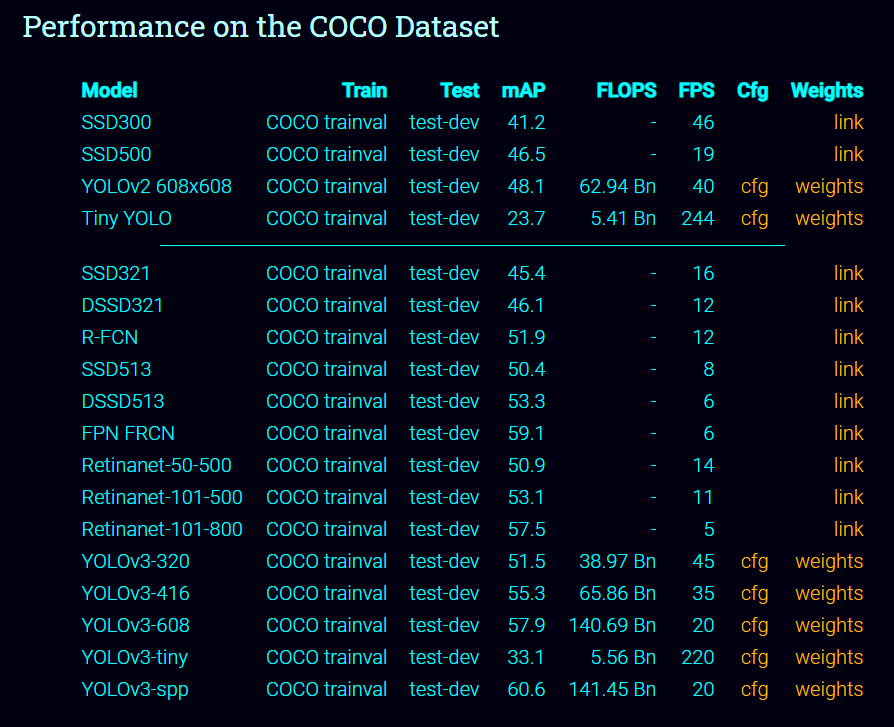
Setelah melewati berbagai operasi *convolution* dan *pooling*, vektor kemudian dilewatkan pada *multilayer* *perceptron* (*fully* *connected* ) untuk melakukan sesuatu (tergantung permasalahan), misal klasifikasi gambar, klasifikasi sentimen, dsb (Ilustrasi pada **Gambar 2.12**).



**Gambar 2.12.**  *Convolutional Neural Network* (Jan Wira, 2019)

## Algoritma YOLO V3 (*You Only Look Once*)

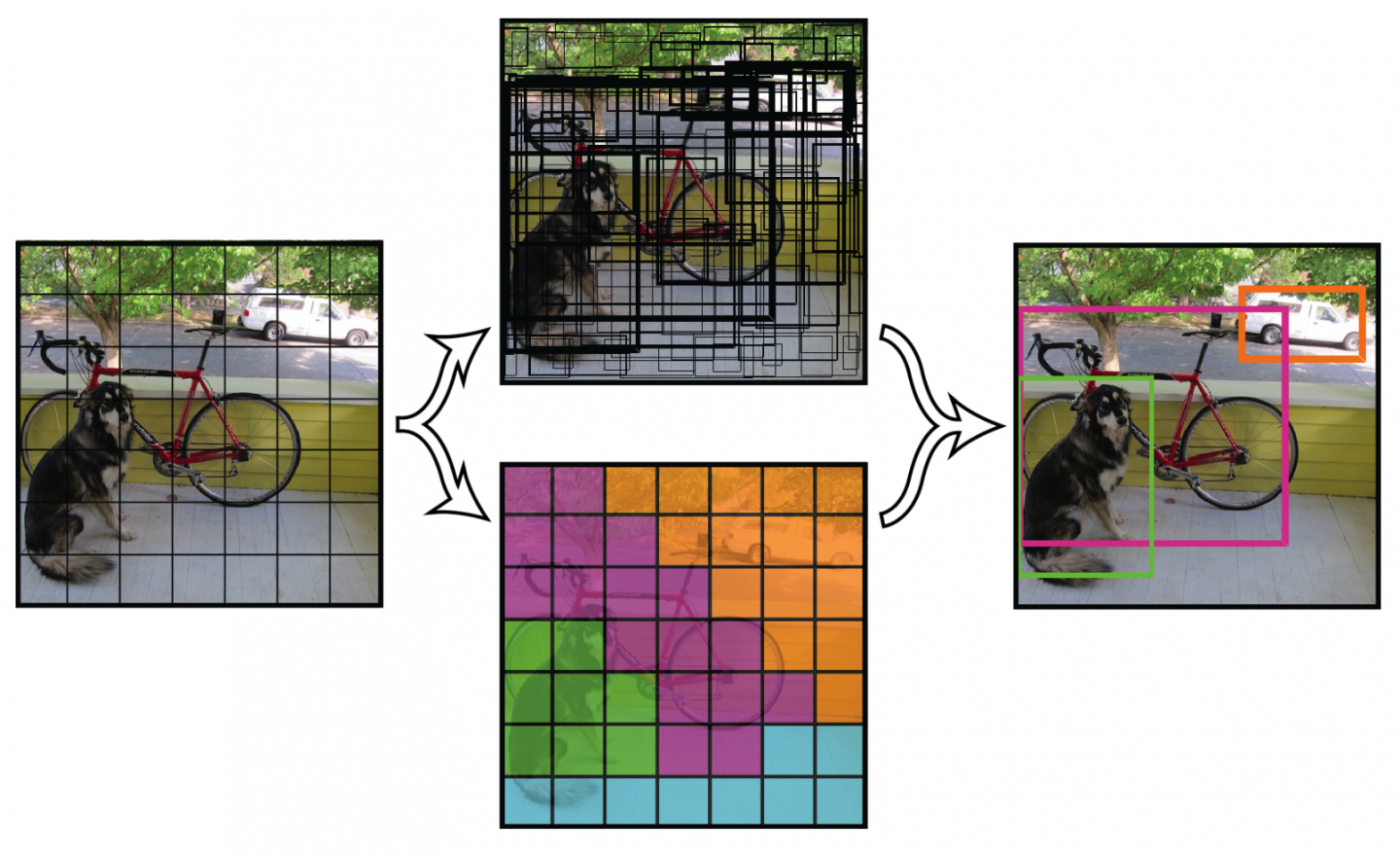
YOLO merupakan sebuah algoritma deep learning yang digunakan untuk mendeteksi objek secara real-time dengan cepat. YOLO mmeiliki 3 versi yaitu YOLO V1 merupakan versi awal YOLO yang sudah jarang digunakan, YOLO V2 yang merupakan versi kedua dari YOLO yang masih banyak digunakan oleh praktisi untuk melakukan pendeteksian secara real-time, dan YOLO V3 yang merupakan versi paling update dan memiliki tingkat akurasi yang tinggi. (joseph, 2018)



**Gambar 2.13.** Perbandingan kinerja deteksi pada *COCO* *Dataset* (pjreddie.com, 2018)

Pada **Gambar 2.13** dapat terlihat bahwa mAp (*mean Average precision*) pada YOLO lebih tinggi dibandingkan algoritma deteksi lain, artinya akurasi yang akan didapat kan jika menggunakan YOLO akan lebih baik jika dibandingkan dengan algoritma lain. (pjredie.com, 2018)

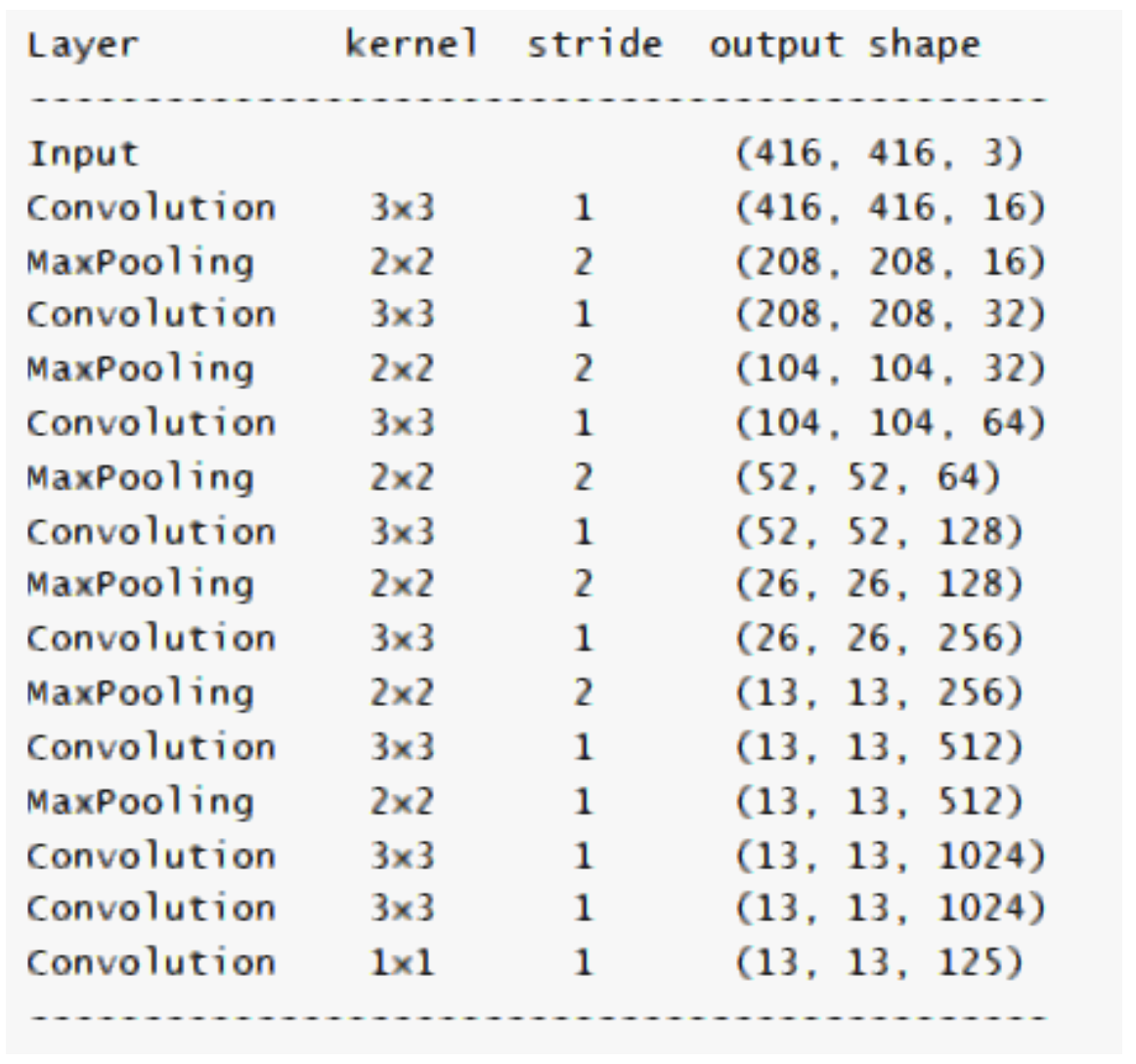
Kebanyakan sistem deteksi sebelumnya menggunakan pengklasifikasian atau *localizer* untuk melakukan deteksi dengan menerapkan model ke gambar di beberapa lokasi dan memberi nilai *confident* pada gambar sebagai bahan untuk pendeteksian. YOLO menggunakan pendekatan yang sangat berbeda dengan algoritma sebelumnya, yakni menerapkan jaringan syaraf tunggal pada keseluruhan gambar. Jaringan ini akan membagi gambar menjadi wilayah-wilayah kemudian memprediksi kotak pembatas dan probabilitas, untuk setiap kotak wilayah pembatas ditimbang probabilitasnya untuk mengklasifikasian sebagai objek atau bukan. Ilustrasi dari proses pendeteksian YOLO dapat dilihat pada **Gambar 2.14.** (Aditya, 2018)



**Gambar 2.14.**  Ilustrasi proses deteksi YOLO (pjreddie.com, 2018)

YOLO memiliki arsitektur yang sederhana yaitu jaringan saraftiruan. Jaringan saraf ini hanya menggunakan jenis lapisan standar: konvolusi dengan kernel 3 × 3 dan *max-pooling* dengan 2 × 2 kernel. Lapisan konvolusional terakhir memiliki 1 × 1 kernel digunakan untuk mengecilkan data ke bentuk 13 × 13 × 125. 13×13 ini seharusnya terlihat familiar karena merupakan ukuran grid yang dibagi menjadi gambar. 125 merupakan Channel untu setiap grid. 125 ini berisi data untuk kotak pembatas dan prediksi kelas. Kenapa 125? setiap sel grid memprediksi 5 kotak sekeliling dan dijelaskan oleh 25 elemen data. (Aditya, 2018)

Ilustrasi dari lapisan konvolusi pada YOLO V3 dapat dilihat pada **Gambar 2.15.**



**Gambar 2.15.** Ilustrasi lapisan konvolusi YOLO V3 (Aditya, 2018)

## Penelitian terkait

Beberapa penelitian telah dilakukan dalam membuat sistem yang dapat mendeteksi dan menghitung sebuah objek. Ibrahim Aliyu , dkk telah melakukan penelitian terkait perhitungan dan pengklasifikasian benih ikan di Nigeria. Mereka menggunakan metode *Image* *processing* dengan melalui beberapa tahapan : *Image* *processing*, *Image* *segmentation*, *Features* *extraction*, *Classification*, *and* *Counting*. Hasil dari penelitian mereka menghasilkan sebuah sistem yang dapat menghitung dan mengklassifikasikan benih ikan namun akurasinya masih harus ditingkatkan.

Kemudian ada Valliappan Raman,dkk melakukan penelitian tentang peranan Machine Learning untuk mendeteksi dan menghitung larva atau *juvenile* ikan di Malaysia. Valliappan menggunakan 4 tahapan untuk mendeteksi dan menghitung jumlah larva dan *juvenile* ikan, Image Acquisition yang dimana menggunakan kamera untuk mengambil gambar dan mengubahnya menjad grayscale, lalu *Image* *Enhancement* yang dimana digunakan untuk menambahkan kontras pada gambar sehingga objek yang ingin di deteksi dapat dilihat dengan jelas, lalu selanjutnya Segmentation yang digunakan untuk mendeteksi *region* *of* *intereset* dari gambar yang memungkinkan sistem untuk mendeteksi kordinat dari tiap sudut *bounding* *box* yang berada di sekeliling objek, dan tahapan terakhir yaitu c*lassification* yang digunakan untuk mengklasifikasikan apakah larva atau *juvenile* saling berdekatan atau tidak sehingga akurasi perhitungannya dapat di tingkatkan. Sistem ini memiliki akurasi sebesar 82% untuk mendeteksi larva ikan dan 87% untuk mendeteksi *juvenile* ikan. Namun sistem ini hanya dapat di terapkan pada gambar yang backgroundnya tidak memiliki *noise*, dan akurasi akan berkurang jikalau pada gambar terdapat banyak *noise*.

Pada tahun 2018, Jirabhorn Kaewchote, Sittichoke Janyong, dan Wasit Limprasert melakukan penelitian terkait perhitungan jumlah anak udang setelah tahapan larva dengan metode *Image* *Processing*. Mereka melakukan pengujian menggunakan 2 metode ekstraksi fitur yaitu RGB dan *Local* *Binary* *Pattern*. Hasil dari pengujiannya ialah metode *Local* *Binary* *Pattern* lebih unggul dengan akurasi 98.50% sedangkan untuk RGB masih salah untuk menghitung jumlah anak udang.

Geoffrey French, Mark Henry Fisher, Michal Mackiewicz, dan Coby Needle melakukan penelitian terkait penggunaan CNN untuk menghitung jumlah ikan pada jarring nelayan melalui CCTV. Mereka melakukan pemrosesan dengan mengekstrak gambar dari video, lalu melakukan pewarnaan kepada ikan yang terdapat pada gambar lalu setelah itu, sudut-sudut dari ikan yang diwarnai di ekstrak untuk dapat melihat *object* yang terdeteksi secara jelas, dan tahapan akhir dengan menggabungkan pewarnaan dengan sudut yang telah di esktrak akan menghasilkan prediksi yang cukup tinggi. Akurasi dari sistem ini dihitung dengan melakukan pengujian pada 6 jaring dan menghasilkan akurasi rata-rata sebesar 90.94%. Berikut adalah **Tabel 2.1**. yang berisi penelitian terkait deteksi dan perhitungan objek dengan metode *machine learning*.

**Tabel 2.1.** Penelitian terkait

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Peneliti | Judul | Objek | Tahun | Metode | Hasil |
| Valliappan Raman, Sundresan Perumal,  Sujata Navaratnam,  Siti Fazilah | *Computer Assisted Counter System for Larvae and Juvenile Fish in Malaysian Fishing Hatcheries by Machine Learning Approach* | Larva dan *Juvenile* Ikan | 2015 | Segmentasi dan Klasifikasi | Akurasi pada Larva 82% dan akurasi pada *Juvenile* 87% |
| Geoffrey French, Mark Henry Fisher, Michal Mackiewicz, Coby Needle | *Convolutional Neural Networks for Counting Fish in Fisheries Surveillance Video* | ikan | 2015 | *Convolutional Neural Network* | Rata- rata akurasi sebesar 90.94% |
| A. Sanchez-Ortiz,  Arista Jlife,  Atoany Nazareth Fierro,  M. Cedillo Hernandez | *Mosquito larva classification method based on convolutional neural networks* | Telur Nyamuk | 2017 | *Convolutional Neural Network* | Akurasi pada sebuah telur nyamuk 96.88% |
| Jirabhorn Kaewchote, Sittichoke Janyong, Wasit Limprasert | *Image recognition method using Local Binary Pattern and the Random forest classifier to count post larvae shrimp* | Larva Udang | 2018 | *Local* *Binary* *Pattern* dan *Random Forest* | Akurasi sistem menggunakan *Local* *Binary* *Pattern* adalah sebesar 98.50% |
| J.H. Almarinez,  Alexander A Hernandez | *Evaluation of mangrove crab classification system* | Larva Kepiting Bakau | 2019 | *K-Nearest Neighbour* | Rata-rata akurasi sebesar 85% |

# BAB III

**METODOLOGI PENELITIAN**

## Tahapan Penelitian

Sistem yang dibuat pada penelitian ini menggunakan algoritma *Deep Learning* YOLO (*You Only Look Once*) versi 3, YOLO menggunakan *Convolutional Neural Network Layer*  untuk melakukan ekstraksi fitur. Untuk menentukan objek larva pada gambar, YOLO melakukan *scoring* sehingga *region* pada gambar yang memiliki *scoring* yang tinggi dapat dipastikan merupakan objek larva pada gambar. Adapun tahapan-tahapan yang dilakukan 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. Tahapan ini di lakukan untuk mengumpulkan penelitian-penelitian terkait metode *Deep Learning* untuk mendeteksi sebuah objek. Pada tahapan ini pula dilakukan penarikan kesimpulan dari penelitian-penelitian terkait.
2. Pengambilan data dilakukan di Balai Perikanan Budidaya Air Payau Takalar. Data diambil menggunakan kamera *smartphone* android dan iOS, gambar juga diambil menggunakan kamera *mirrorless* untuk menghasilkan gambar dengan resolusi yang baik.
3. *Preprocessing* merupakan tahapan untuk melakukan penyortiran data untuk menyeleksi data yang resolusinya dibawah standar, setelah di sortir, data akan di beri label menggunakan program YOLO\_mark untuk menandai objek yang ingin di deteksi.
4. Metode yang digunakan untuk membuat sistem pada penelitian ini merupakan metode *Object Detection* pada *Deep Learning* dengan algoritma YOLO (*You Only Look Once*) versi ketiga.
5. *Processing* merupakan tahapan untuk melakukan pelatihan dari data yang telah diberi label, pada tahap ini pelatihan dilakukan menggunakan *Google Colaboratory. Google Colaboratory* merupakan sebuah *service* dari *Google* yang dikhususkan untuk pengembang *project* *Machine Learning*.
6. Implementasi dari Sistem dibuat menggunakan bahasa pemrograman Python dengan *framework* darknet.
7. Uji coba sistem dilakukan dengan melakukan pengujian sistem untuk mendeteksi larva pada gambar-gambar yang diberikan sehingga akurasi dari sistem dapat di lihat, pada tahapan ini juga diukur waktu yang digunakan oleh program untuk memproses 1 gambar hingga 200 gambar.
8. Tahapan akhir dalam penelitian ini adalah dengan melakukan penulisan laporan penelitian dalam bentuk skripsi sebagai bahan publikasi.

## **Waktu** **dan** **Lokasi** **Penelitian**

Penelitian ini telah dilakukan selama kurang lebih 5 bulan dimulai pada bulan Juni 2019 hingga bulan ditulisnya laporan ini yaitu bulan Oktober 2019. Pengambilan data pada penelitian ini dilakukan di Balai Perikanan Budidaya Air Payau Takalar. Sedangkan untuk pengolahan data di lakukan di laboratorium UBICON fakultas Teknik universitas hasanuddin.

## Instrumen Penelitian

Instrumen penelitian yang digunakan pada penelitian ini meliputi :

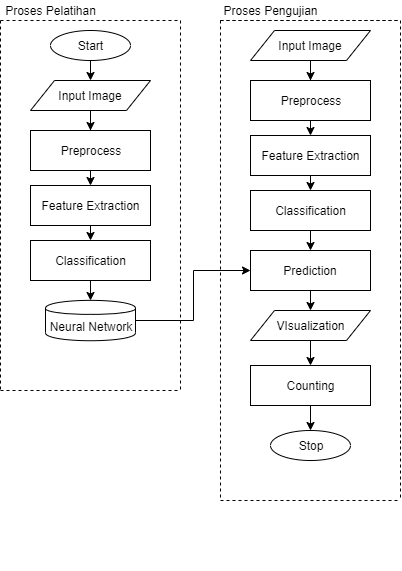
1. Software
2. Windows 10 Home 64-bit
3. Microsoft Visual Studio 2017 Community
4. Jupyter Notebook Environment
5. Google Colaboratory (Intel Xeon [CPU@2.20GHz](mailto:CPU@2.20GHz) NVIDIA Tesla K80, 12 GB GDDR5VRAM)
6. Hardware
7. Laptop Acer Aspire A715-71G, RAM 8GB, Prosesor Intel(R) Core(TM) i5-7300HQ CPU @2.50GHz
8. *Smartphone* Xiaomi Redmi A4
9. *Smartphone* iPhone 5
10. Kamera Sony RX 100

## Teknik Pengambilan Data

Data yang digunakan pada penelitian ini menggunakan gambar yang diambil secara langsung menggunakan kamera *smartphone* serta kamera *digital.* Larva yang akan diambil gambarnya dipindahkan di sebuah wadah dengan air yang cukup dan berisi 5 sampai 10 larva. Larva pada penelitian ini merupakan larva dengan tahapan *Zoea 1* hingga *Zoea 4.* Gambar yang telah dikumpulkan kemudian di saring untuk mendapatkan gambar dengan resolusi terbaik yang akan digunakan untuk sebagi data *training*.

## Perancangan Sistem

Secara umum sistem terbagi menjadi dua proses, yaitu proses pelatihan dan proses pengujian seperti yang dapat dilihat pada **Gambar 3.2.** Pada proses pelatihan, fitur yang telah di ekstrak dari data *training* menggunakan layer *Convolutional Neural Network* akan dijadikan input ke dalam *Fully Connected Layer* yang akan menghasilkan model deteksi larva. Model yang dihasilkan oleh proses pelatihan ini akan di gunakan untuk proses pengujian sistem.



**Gambar 3.2** *Flowchart*  perancangan sistem deteksi dan perhitungan larva rajungan

Berdasarkan **Gambar 3.2**, perancangan sistem deteksi dan perhitungan larva rajungan dapat diuraikan sebagai berikut.



### *Input Image*

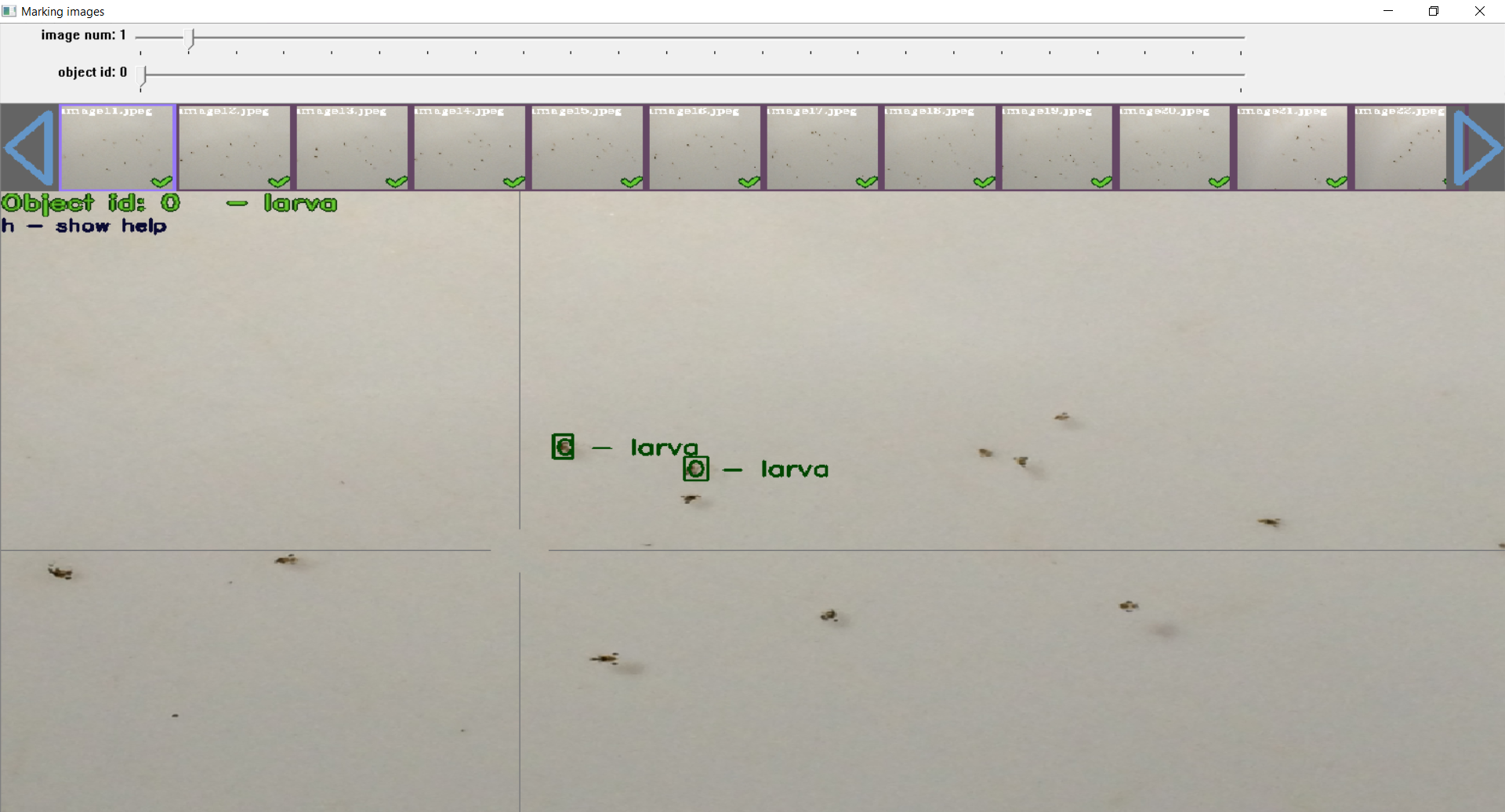
Langkah pertama ialah dengan mempersiapkan gambar larva kepiting rajungan yang akan digunakan untuk pelatihan dan pengujian. Dalam penelitian ini, gambar diambil menggunakan kamera *smartphone* dan kamera *digital,* selain itu data gambar juga di peroleh dari hasil ekstraksi gambar dari video yang diambil menggunakan kamera *smartphone.* Gambar di esktrak setiap 1 detik dari video menggunakan library python yang bernama FFMPEG. Berikut adalah contoh gambar yang diambil untuk dijadikan data latih seperti yang terlihat pada **Gambar 3.3.**



**Gambar 3.3** Contoh gambar masukan

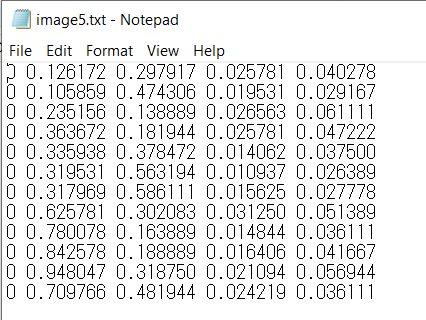
Gambar yang telah terkumpul akan di beri label secara manual menggunakan program YOLO *Mark*. Langkah-langkah memberi label adalah sebagai berikut :

1. *Edit* file obj.names, file tersebut berisi nama *class* yang ingin dideteksi oleh sistem. Nama *class* satu dan nama *class* lainnya dipisahkan dengan baris baru, *id* *class* dimulai dari 0 pada baris pertama.
2. *Edit* file obj.data, nilai pada variabel *class* akan di ganti dengan jumlah *class* yang akan dideteksi yang dalam hal ini adalah 1.
3. Data yang akan di beri label akan di masukkan kedalam file “img”.
4. Membuka Yolo\_mark.exe dan menggambar kotak disekeliling objek pada gambar. Setelah menggambar kotak di sekeliling objek, secara otomatis program akan memunculkan nama *class* serta *id class*. *Infterface* dari progam dapat dilihat pada **Gambar 3.4.**



**Gambar 3.4.** *Interface* dari program *YOLO* *Mark*

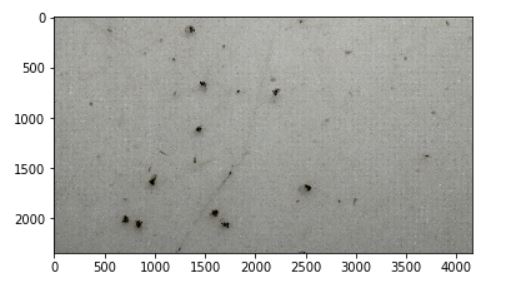
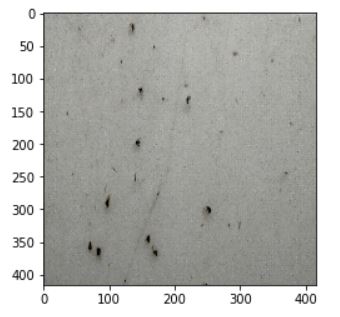
1. Setiap gambar yang telah diberi label, secara otomati program *YOLO Mark* akan menghasilkan sebuah file anotasi dalam format *text* yang berisi kordinat dan *id class*  dengan format . Contoh dari file anotasi yang dihasilkan oleh program *YOLO Mark* dapat dilihat pada **Gambar 3.5.**



**Gambar 3.5.** Contoh file anotasi dari program *YOLO Mark*

### *Preprocess*

YOLO membutuhkan gambar masukan dengan resolusi 416x416 karena resolusi ini merupakan resolusi optimal untuk dapat mendeteksi objek, bahkan untuk mendeteksi objek berukuran kecil sekalipun. Nilai resolusi ini juga digunakan untuk mengantisipasi *eror* pada saat melakukan training dengan jumlah data gambar yang sangat banyak dan memiliki resolusi yang bervariasi. Maka dari itu setiap gambar yang dimasukkan ke *neural network* YOLO akan di *resize* resolusinya menjadi 416x416 piksel. Ilustrasi proses *resize* gambar dapat dilihat pada **Gambar 3.6.**

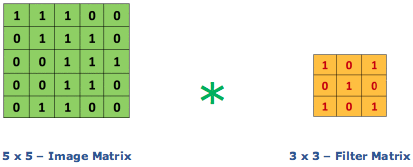
 

(a) (b)

**Gambar 3.6** (a) gambar sebelum di *resize,* (b) gambar setenlah di *resize*

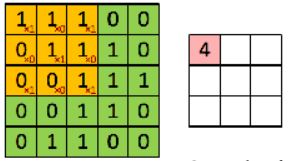
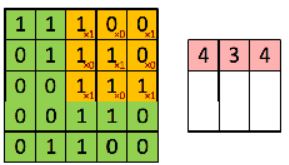
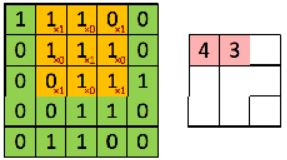
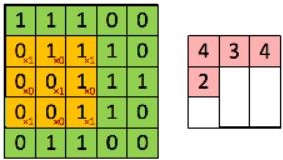
### *Feature Extraction*

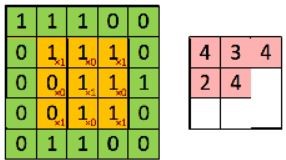
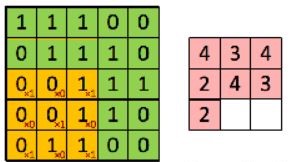
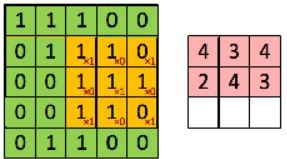
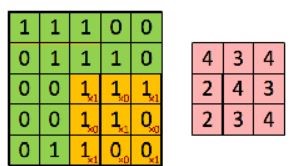
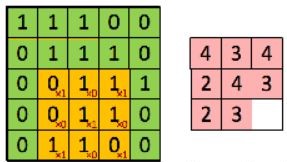
Pada model *Convolutional Neural Network* , setiap gambar masukan akan melalui beberapa tahapan *convolutional layer* dengan *filter*, *Pooling layer,* dan *Fully connected layer.* Layer pertama untuk mengekstrak fitur adalah *Convolutional layer*, layer ini menjaga hubungan antar piksel pada gambar dengan menggunakan kotak-kotak kecil sebagai data masukan. Hal ini menggunakan operasi matematika yang dimana mengambil 2 masukan yang dalam hal ini adalah matriks dari gambar dan sebuah *filter* atau *kernel.* Perkalian matriks dapat dilihat pada **Gambar 3.7** yang dimana *image matrix* memiliki jumlah pixel 5x5 dan memiliki nilai pixel 0 dan 1. Kemudian matriks gambarakan dioperasikan dengan matriks filter dengan jumlah pixel 3x3.



**Gambar 3.7** Perkalian matriks gambar dan matriks filter

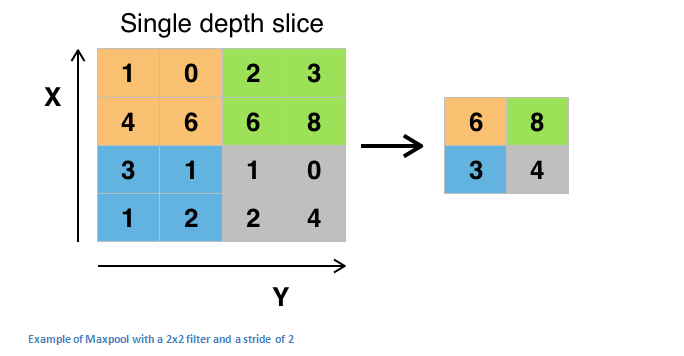
Operasi dari matriks tersebut menghasilkan *output* berupa sebuah konvolusi yang disebut *Feature Map* yang dapat dilihat pada **Gambar 3.8.**

****  

**Gambar 3.8.** Contoh perkalian sehingga menghasilkan*Feature Map*

Setelah itu *filter* akan bergeser pada gambar sesuai dengan jumlah *stride. Stride* adalah sebuah parameter filter yang ditentukan untuk menentukan jumlah pregeseran piksel pada gambar. Contohnya jika nilai *stride* adalah 1 maka *filter* akan bergeser tiap 1 pixel dalam suatu waktu, ilustrasi dari proses perpindahan dapat *stride* dapat dilihat pada **Gambar 3.9.**

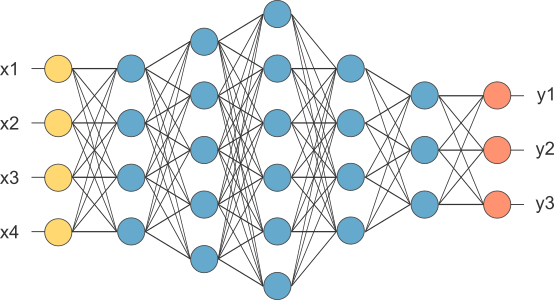


**Gambar 3.9** Pergeseran *filter* dengan *stride*=2

Dalam sistem yang dibuat terdapat *Pooling layer* yang berfungsi untuk mengurangi jumlah parameter yang digunakan saat gambar terlalu besar, ini diperlukan untuk mempercepat proses komputasi model. Dalam sistem yang digunakan ialah tipe pooling berjenis *Max Poolin*g yang berguna untuk mengurangi dimensi dari tiap map tanpa kehilangan informasi-informasi penting.

### *Classification*

Matriks hasil dari *Max pooling layer* akan di *reshape* menjadi bentuk vektor yang kemudian di masukkan ke dalam *Fully Connected Layer.* Dalam tahapan ini model akan di klasifikasikan untuk mengenali apakah objek yang terdapat pada gambar merupakan larva atau bukan, jenis klasifikasi ini merupakan *Binary Classification*. Ilustrasi dari *Fully Connected Layer* ini dapat dilihat pada **Gambar 3.10** yang dimana *feature map* akan di konversi menjadi bentuk vector *(x1,x2,x3,…).* Dengan *Fully Connected Layer*, fitur akan di gabungkan untuk membentuk sebuah model. Gabungan dari fitur-fitur disebut akan membentuk sebuah *activation map* yang akan mengklasifikasikan larva atau bukan larva.



**Gambar 3.10.** *Fully Connected Layer*

### *Visualization*

Untuk menghasilkan gambar yang telah di proses oleh sistem, digunakan OpenCV, yang dimana berfungsi untuk mempercepat proses komputasi data dan untuk menggambar *bounding box*. YOLO memprediksi sebuah object menggunakan *objectness score* pada setiap *bounding box* untuk menentukan apakah *bounding box* akan di gambarkan atau tidak. Secara default nilai threshold untuk *objectness score* adalah 0.5. Jika nilai *objectness* *score* pada sebuah *bounding* *box* terpenuhi maka sistem akan menggambar *bounding* *box* sesuai dengan nilai pada persamaan 3.1 yang dimana *bx* dan *by* merupakan kordinat titik tengah *x* dan *y*, kemudian *bw* dan *bh* merupakan *width* dan *height* dari prediksi. *tx,ty,tw,th* merupakan keluaran dari proses training. Sedangkan *cx* dan *cy* merupakan kordinat ujung kiri dari *grid* pada *bounding box*. *pw* dan *ph* adalah dimensi *anchor* untuk *bounding box.*

(3.1)

### *Counting*

Proses perhitungan dilakukan dengan mendeteksi jumlah *bounding box* yang telah divisualisasikan pada gambar*. Bounding box* yang telah di gambarkan akan di cocokkan *id* nya dengan *id* *class* yang ingin di hitung jumlahnya, jika *id* dari *bounding box* sama dengan *id class* maka jumlah larva akan bertambah 1. Total keseluruhan dari gambar larva yang di proses oleh sistem akan di jumlahkan dan ditampilkan di akhir pemrosesan sistem yang juga disimpan pada sebuah file txt.

## Analisis kinerja sistem

Analisis kinerja sistem deteksi dan perhitungan larva kepiting rajungan dapat dilakukan dengan menghitung nilai akurasi menggunakan *confussion matrix.* Nilai akurasi menggambarkan seberapa akurat sistem dapat mengklasifikasikan data secara benar. Dengan kata lain, nilai akurasi merupakan perbandingan antara data yang terklasifikasi benar dengan keseluruhan data. *Confussion Matrix* yang digunakan dapat dilihat pada **Tabel 3.1.**

**Tabel 3.1** *Confussion Matrix*

|  |  |  |
| --- | --- | --- |
| **Kelas** | **Terklasifikasi Positif** | **Terklasifikasi Negatif** |
| **Positif** | *TP (True Positive)* | *FP (False Positive)* |
| **Negatif** | *FN (False Negative)* | *TN (True Negative)* |

Keterangan dari *Confussion Matrix* adalah sebagai berikut :

* *True* *Positive* : Jumlah data positif yang terklasifikasi benar oleh sistem
* *False* *Positive* : Jumlah data positif yang terklasifikasi salah oleh sistem
* *False* *Negative* : Jumlah data negatif yang terklasifikasi salah oleh sistem
* *True* *Negative* : Jumlah data negatif yang terklasifikasi benar oleh sistem

Untuk menghitung nilai akurasi, dapat dilakukan menggunakan persamaan 3.2 berikut.

(3.2)

# BAB IV

**HASIL DAN PEMBAHASAN**

## Hasil Penelitian

Bab ini menjelaskan terkait hasil dari kinerja sistem deteksi dan perhitungan larva kepiting rajungan dengan metode *object detection* yang dalam hal ini menggunakan *Convolutional Neural Network* dengan algoritma YOLO. Pada penelitian ini menggunakan *Confussion Matrix* untuk evaluasi kinerja sistem, total keseluruhan gambar yang digunakan untuk melakukan *training* ialah 375 gambar larva yang diambil dari berbagai resolusi dan di bagi menjadi 80% data *training* dan 20% data *test* atau sekitar 300 data *training* dan 75 data *testing*, contoh gambar yang dijadikan *training* set dapat dilihat pada **Gambar 4.1.**



**Gambar 4.1** Contoh gambar larva untuk data training

Parameter yang digunakan pada penelitian ini ialah *batch*, *subdivision*, *width*, *height*, *channels*, *learning rate*, *steps, scale, max\_batches*, *data augmentation* (perubahan *angle* , *saturation*, *exposure*, serta *hue* pada gambar data *training*). Nilai yang digunakan oleh parameter-parameter di atas merupakan nilai *default* seperti berikut :

* Batch = 64
* Subdivisions = 32
* Width = 416
* Height = 416
* Channel = 3
* Learning rate = 0.001
* Steps = 1600, 1800
* Scale = 0.1, 0.1
* Max\_batches = 2000

*Training* awal dilakukan sebanyak 2000 iterasi, kemudian ditambah lagi sebanyak 2000 iterasi agar dapat mengevaluasi nilai akurasi dari file bobot yang dihasilkan. **Tabel 4.1** menunjukkan hasil dari evaluasi mAP sistem menggunakan *confussion matrix.*

**Tabel 4.1.** Evaluasi nilai mAP sistem menggunakan *confussion matrix*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Jumlah Iterasi | mAP | *True Positive* | *False Negative* | *False Positive* | *True Negative* |
| 1000 | 70.12% | 419 | 202 | 96 | 0 |
| 2000 | 86.16% | 553 | 68 | 75 | 0 |
| 3000 | 63.73% | 367 | 49 | 254 | 0 |
| 4000 | 64.14% | 389 | 232 | 44 | 0 |

Sistem dengan iterasi sebanyak 2000 merupakan yang paling baik akurasinya, sehingga sistem dengan 2000 iterasi kemudian di uji coba pada 10 gambar larva yang memiliki ukuran resolusi yang berbeda, jumlah larva yang berbeda dan objek larva dengan tahapan yang berbeda. Hasil dari Pengujian sistem dapat dilihat pada **Tabel 4.2**

**Tabel 4.2.** Hasil pengujian sistem

|  |  |  |
| --- | --- | --- |
| Data | Jumlah larva pada gambar | Jumlah larva yang terdeteksi |
| Data 1 | 13 larva | 11 larva |
| Data 2 | 42 larva | 40 larva |
| Data 3 | 53 larva | 45 larva |
| Data 4 | 11 larva | 9 larva |
| Data 5 | 79 larva | 72 larva |

## Pembahasan

Berikut penjelasan dari parameter-parameter yang digunakan untuk membuat sistem pada penelitian ini



### *Batch*

Parameter ini merupakan sebuah nilai yang menunjukkan berapa banyak data yang akan di proses oleh 1 waktu dan disebar ke *neural network*. Dalam hal ini nilai yang di gunakan adalah 64 yang artinya sistem akan mengambil 64 gambar pertama dari *training set* setiap waktu dan akan di proses di *neural network* kemudian setelah selesai diproses, sistem akan mengambil 64 gambar kedua dan dilakukan terus menerus hingga habis dan diulang lagi sesuai jumlah iterasi. Mengapa hal ini dilakukan?? Ini dikarenakan komputer tidak dapat memproses data yang banyak dalam satu waktu sehingga dibutuhkan penyebaran data agar computer dapat melakukan training.

### *Max Batches (Iteration)*

Parameter ini menunjukkan berapa banyak pengulangan yang akan dilakukan oleh sistem selama training berlangsung hingga file bobot dapat di hasilkan. Tidak ada nilai pasti untuk parameter ini, namun referensi dari jumlah parameter *Max Batches* dapat ditentukan dengan rumus 2000\* jumlah kelas yang ingin dideteksi. Karena jumlah kelas yang ingin dideteksi pada penelitian ini hanya 1 yaitu larva, maka nilai parameter ini adalah 2000.

### *Subdivision*

Parameter ini menunjukkan nilai yang digunakan untuk membagi nilai batch, walaupun gambar yang diproses oleh sistem sudah dibatasi dengan menggunakan parameter batch size yang bernilai 64, tetap saja komputer akan kesulitan untuk memproses data sehingga akan muncul eror pada GPU yang disebabkan oleh penggunaan memori yang berlebihan. Maka dari itu data sebanyak 64 ini akan di bagi kembali dengan memasukkan nilai subdivision sebesar 32 maka sistem akan memproses 2 gambar dalam 1 waktu.

### *Width, Height dan Channel*

Parameter-parameter ini menunjukkan ukuran dari gambar yang digunakan untuk melakukan pelatihan. Nilai dari parameter *Width* dan *Height* adalah nilai resolusi gambar saat akan di *resize* ke resolusi 416x416 piksel. Nilai parameter *Channel* adalah 3 yang artinya gambar yang digunakan untuk melakukan pelatihan adalah gambar dengan 3 channel yaitu RGB (*Red*, *Green*, *Blue*)

### *Learning Rate, Steps,* dan *Scale*

Parameter *learning* *rate* menunjukkan sebarapa agresif sistem akan melakukan pembelajaran saat melakukan pelatihan. Nilai yang di terapkan ialah 0.001 (biasanya bernilai 0.001-0.0001), namun selama pelatihan berjalan dan sistem mengenali semakin banyak gambar maka nilai *learning* *rate* akan semakin berkurang. Nilai *steps* yang digunakan adalah 1600 dan 1800 yang artinya salama 1600 iterasi, nilai *learning rate* akan konstan dan setelah itu akan dikalikan dengan nilai *scale* yang dalam hal ini adalah 0.1.

### *Data Augmentation (angle, saturation, exposure, hue)*

Saat melakukan pelatihan, sistem membutuhkan banyak data *training*, namun untuk melakukan hal itu tentunya dibutuhkan usaha yang berlebih untuk mengumpulkan gambar lalu melabeli gambar satu persatu. Data augmentation berperan dalam hal menduplikasi data *training* yang telah dikumpulkan lalu memberikan sedikit perubahan seperti mengubah sudut gambar dengan melakukan rotasi, merubah warna dengan mengkonfigurasi *saturation*, *exposure*, dan *hue*.

# BAB V

**PENUTUP**

## Kesimpulan

Berdasarkan hasil pengujian dari implementasi sistem deteksi dan perhitungan lava kepiting rajungan dengan metode *object detection* yang kemudian di analisa dan disimpulkan bahwa :

1. Sistem deteksi dan perhitungan larva kepiting rajungan ini menggunakan algoritma deep learning YOLO V3 untuk mendeteksi objek. Gambar di ambil menggunakan kamera dan dikumpulkan lalu diberi label secara manual. Gambar yang telah di beri label akan di latih untuk menghasilkan sebuah file bobot. Metode yang digunakan untuk mengekstraksi fitur dan kalsifikasi adalah *Convolutional Neural Network* yang didalamnya terdapat 106 layer. Hasil dari deteksi program akan di visualisasi dengan secara otomatis menggambarkan *bounding box* di sekitar objek yang terdeteksi pada gambar. *Bounding box* kemudian akan dihitung untuk dapat menentukan hasil dari seluruh jumlah larva pada gambar.
2. Sistem menghasilkan nilai akurasi sebesar 86.16% dengan nilai parameter *batch* 64, *subdivision* 32, *width* 416, *height* 416, *channels* 3, *learning rate* 0.001, *steps* 1600 dan 1800*, scale* 0.1*,*serta *max\_batches* atau jumlah iterasinya mencapai 2000.

## Saran

Saran dari penulis untuk pengembangan sistem ini yang dapat dilakukan kedepannya adalah sebagai berikut :

1. Sistem yang telah dibuat masih dapat ditingkatkan akurasinya dengan menambahkan data training hingga mencapai 2000 gambar/*class*. File konfigurasi YOLO juga dapat di ubah untuk meningkatkan akurasi, seperti merubah nilai parameter *width* dan *height*, *max* *batch*, *layer*, *stride*, dan nilai *learning rate*.
2. Sistem ini dapat diimplementasi dalam berbagai bidang sehingga diharaokan tidak hanya untuk bidang *aquaculture*, tapi sistem ini dapat diterapkan di bidang lain peternakan, pertanian, kedokteran,dll dengan mengubah data *training* nya.
3. Framework yang digunakan pada sistem ini adalah darknet, namun masih ada framework lain seperti *Tensorflow* atau *Darkflow* (gabungan dari *darknet* dan *tensorflow*) yang dapat digunakan untuk mempermudah pemrosesan dari sistem.

# DAFTAR PUSTAKA

Aliyu, I., Gana, K. J., Musa, A. A., Agajo, J., Orire, A. M., Abiodun, F. T., & Adegboye, M. A. (2017). A PROPOSED FISH COUNTING ALGORITHM USING DIGITAL IMAGE PROCESSING. *Journal of Science, Technology & Education (JOSTE), V*.

Almarinez, J. H., & Hernandez, A. (2019). Evaluation of Mangrove Crab Classification System. *International Journal of Recent Technology and Engineering (IJRTE)*, 2277-3878.

Fierro, A. N., Camarillo, D. R., & Miyatake, M. N. (2017). Mosquito larva classification method based on convolutional neural networks. *International Conference on Electronics, Communications and Computers (CONIELECOMP)*.

French, G., Fisher, M. H., Mackiewicz, M., & Needle, C. (2015). Convolutional Neural Networks for Counting Fish in Fisheries Surveillance Video. *Workshop on Machine Vision of Animals and their Behaviour*.

Kaewchote, J., Janyong, S., & Limprasert, W. (2018). Image recognition method using Local Binary Pattern and the Random forest classifier to count post larvae shrimp. *Agriculture and Natural Resources*.

Munir, R. (2004). *Pengolahan Citra Digital Dengan Pendekatan Algoritmik.* Bandung: INFORMATIKA.

Putra, J. G. (2019). *Pengenalan Konsep Pembelajaran Mesin dan Deep Learning.* Tokyo.

Raman, V., Perumal, S., Navaratnam, S., & Fazilah, S. (2015). Computer Assisted Counter System for Larvae and Juvenile Fish in Malaysian Fishing Hatcheries by Machine Learning Approach. *Journal of Computers*, 423-431.

Redmon, J., & Farhadi, A. (2018). YOLOV3: An Incremental Impovement. *arXiv*.

Redmon, J., Vala, S. D., Girshick, R., & Farhadi, A. (2016). You Only Look Once: Unified, Real-Time Object Detection. *arXiv*.

# LAMPIRAN

**Lampiran 1**. *Source* *code* Makefile untuk *compile* darknet

GPU=1

CUDNN=1

CUDNN\_HALF=0

OPENCV=1

AVX=0

OPENMP=0

LIBSO=0

ZED\_CAMERA=0

# set GPU=1 and CUDNN=1 to speedup on GPU

# set CUDNN\_HALF=1 to further speedup 3 x times (Mixed-precision on Tensor Cores) GPU: Volta, Xavier, Turing and higher

# set AVX=1 and OPENMP=1 to speedup on CPU (if error occurs then set AVX=0)

DEBUG=0

ARCH= -gencode arch=compute\_30,code=sm\_30 \

-gencode arch=compute\_35,code=sm\_35 \

-gencode arch=compute\_50,code=[sm\_50,compute\_50] \

-gencode arch=compute\_52,code=[sm\_52,compute\_52] \

-gencode arch=compute\_61,code=[sm\_61,compute\_61]

OS := $(shell uname)

VPATH=./src/

EXEC=darknet

OBJDIR=./obj/

ifeq ($(LIBSO), 1)

LIBNAMESO=libdarknet.so

APPNAMESO=uselib

endif

CC=gcc

CPP=g++

NVCC=nvcc

OPTS=-Ofast

LDFLAGS= -lm -pthread

COMMON= -Iinclude/ -I3rdparty/stb/include

CFLAGS=-Wall -Wfatal-errors -Wno-unused-result -Wno-unknown-pragmas -fPIC

ifeq ($(DEBUG), 1)

#OPTS= -O0 -g

#OPTS= -Og -g

COMMON+= -DDEBUG

CFLAGS+= -DDEBUG

else

ifeq ($(AVX), 1)

CFLAGS+= -ffp-contract=fast -mavx -mavx2 -msse3 -msse4.1 -msse4.2 -msse4a

endif

endif

CFLAGS+=$(OPTS)

ifeq ($(OPENCV), 1)

COMMON+= -DOPENCV

CFLAGS+= -DOPENCV

LDFLAGS+= `pkg-config --libs opencv`

COMMON+= `pkg-config --cflags opencv`

endif

ifeq ($(OPENMP), 1)

CFLAGS+= -fopenmp

LDFLAGS+= -lgomp

endif

ifeq ($(GPU), 1)

COMMON+= -DGPU -I/usr/local/cuda/include/

CFLAGS+= -DGPU

ifeq ($(OS),Darwin) #MAC

LDFLAGS+= -L/usr/local/cuda/lib -lcuda -lcudart -lcublas -lcurand

else

LDFLAGS+= -L/usr/local/cuda/lib64 -lcuda -lcudart -lcublas -lcurand

endif

endif

ifeq ($(CUDNN), 1)

COMMON+= -DCUDNN

ifeq ($(OS),Darwin) #MAC

CFLAGS+= -DCUDNN -I/usr/local/cuda/include

LDFLAGS+= -L/usr/local/cuda/lib -lcudnn

else

CFLAGS+= -DCUDNN -I/usr/local/cudnn/include

LDFLAGS+= -L/usr/local/cudnn/lib64 -lcudnn

endif

endif

ifeq ($(CUDNN\_HALF), 1)

COMMON+= -DCUDNN\_HALF

CFLAGS+= -DCUDNN\_HALF

ARCH+= -gencode arch=compute\_70,code=[sm\_70,compute\_70]

endif

ifeq ($(ZED\_CAMERA), 1)

CFLAGS+= -DZED\_STEREO -I/usr/local/zed/include

LDFLAGS+= -L/usr/local/zed/lib -lsl\_core -lsl\_input -lsl\_zed

#-lstdc++ -D\_GLIBCXX\_USE\_CXX11\_ABI=0

endif

OBJ=image\_opencv.o http\_stream.o gemm.o utils.o dark\_cuda.o convolutional\_layer.o list.o image.o activations.o im2col.o col2im.o blas.o crop\_layer.o dropout\_layer.o maxpool\_layer.o softmax\_layer.o data.o matrix.o network.o connected\_layer.o cost\_layer.o parser.o option\_list.o darknet.o detection\_layer.o captcha.o route\_layer.o writing.o box.o nightmare.o normalization\_layer.o avgpool\_layer.o coco.o dice.o yolo.o detector.o layer.o compare.o classifier.o local\_layer.o swag.o shortcut\_layer.o activation\_layer.o rnn\_layer.o gru\_layer.o rnn.o rnn\_vid.o crnn\_layer.o demo.o tag.o cifar.o go.o batchnorm\_layer.o art.o region\_layer.o reorg\_layer.o reorg\_old\_layer.o super.o voxel.o tree.o yolo\_layer.o upsample\_layer.o lstm\_layer.o conv\_lstm\_layer.o

ifeq ($(GPU), 1)

LDFLAGS+= -lstdc++

OBJ+=convolutional\_kernels.o activation\_kernels.o im2col\_kernels.o col2im\_kernels.o blas\_kernels.o crop\_layer\_kernels.o dropout\_layer\_kernels.o maxpool\_layer\_kernels.o network\_kernels.o avgpool\_layer\_kernels.o

endif

OBJS = $(addprefix $(OBJDIR), $(OBJ))

DEPS = $(wildcard src/\*.h) Makefile include/darknet.h

all: obj backup results setchmod $(EXEC) $(LIBNAMESO) $(APPNAMESO)

ifeq ($(LIBSO), 1)

CFLAGS+= -fPIC

$(LIBNAMESO): $(OBJS) include/yolo\_v2\_class.hpp src/yolo\_v2\_class.cpp

$(CPP) -shared -std=c++11 -fvisibility=hidden -DLIB\_EXPORTS $(COMMON) $(CFLAGS) $(OBJS) src/yolo\_v2\_class.cpp -o $@ $(LDFLAGS)

$(APPNAMESO): $(LIBNAMESO) include/yolo\_v2\_class.hpp src/yolo\_console\_dll.cpp

$(CPP) -std=c++11 $(COMMON) $(CFLAGS) -o $@ src/yolo\_console\_dll.cpp $(LDFLAGS) -L ./ -l:$(LIBNAMESO)

endif

$(EXEC): $(OBJS)

$(CPP) -std=c++11 $(COMMON) $(CFLAGS) $^ -o $@ $(LDFLAGS)

$(OBJDIR)%.o: %.c $(DEPS)

$(CC) $(COMMON) $(CFLAGS) -c $< -o $@

$(OBJDIR)%.o: %.cpp $(DEPS)

$(CPP) -std=c++11 $(COMMON) $(CFLAGS) -c $< -o $@

$(OBJDIR)%.o: %.cu $(DEPS)

$(NVCC) $(ARCH) $(COMMON) --compiler-options "$(CFLAGS)" -c $< -o $@

obj:

mkdir -p obj

backup:

mkdir -p backup

results:

mkdir -p results

setchmod:

chmod +x \*.sh

.PHONY: clean

clean:

rm -rf $(OBJS) $(EXEC) $(LIBNAMESO) $(APPNAMESO)

**Lampiran 2**. *Source code* YOLO V3 dengan framework 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"

@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)

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)

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, 0) # OpenCV

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

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 3**. *Source code* konfigurasi *training* YOLO V3

[net]

# Testing

#batch=1

#subdivisions=1

# Training

batch=64

subdivisions=32

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 = 2000

policy=steps

steps=1600,1800

scales=.1,.1

[convolutional]

batch\_normalize=1

filters=32

size=3

stride=1

pad=1

activation=leaky

# Downsample

[convolutional]

batch\_normalize=1

filters=64

size=3

stride=2

pad=1

activation=leaky

[convolutional]

batch\_normalize=1

filters=32

size=1

stride=1

pad=1

activation=leaky

[convolutional]

batch\_normalize=1

filters=64

size=3

stride=1

pad=1

activation=leaky

[shortcut]

from=-3

activation=linear

# Downsample

[convolutional]

batch\_normalize=1

filters=128

size=3

stride=2

pad=1

activation=leaky

[convolutional]

batch\_normalize=1

filters=64

size=1

stride=1

pad=1

activation=leaky

[convolutional]

batch\_normalize=1

filters=128

size=3

stride=1

pad=1

activation=leaky

[shortcut]

from=-3

activation=linear

[convolutional]

batch\_normalize=1

filters=64

size=1

stride=1

pad=1

activation=leaky

[convolutional]

batch\_normalize=1

filters=128

size=3

stride=1

pad=1

activation=leaky

[shortcut]

from=-3

activation=linear

# Downsample

[convolutional]

batch\_normalize=1

filters=256

size=3

stride=2

pad=1

activation=leaky

[convolutional]

batch\_normalize=1

filters=128

size=1

stride=1

pad=1

activation=leaky

[convolutional]

batch\_normalize=1

filters=256

size=3

stride=1

pad=1

activation=leaky

[shortcut]

from=-3

activation=linear

[convolutional]

batch\_normalize=1

filters=128

size=1

stride=1

pad=1

activation=leaky

[convolutional]

batch\_normalize=1

filters=256

size=3

stride=1

pad=1

activation=leaky

[shortcut]

from=-3

activation=linear

[convolutional]

batch\_normalize=1

filters=128

size=1

stride=1

pad=1

activation=leaky

[convolutional]

batch\_normalize=1

filters=256

size=3

stride=1

pad=1

activation=leaky

[shortcut]

from=-3

activation=linear

[convolutional]

batch\_normalize=1

filters=128

size=1

stride=1

pad=1

activation=leaky

[convolutional]

batch\_normalize=1

filters=256

size=3

stride=1

pad=1

activation=leaky

[shortcut]

from=-3

activation=linear

[convolutional]

batch\_normalize=1

filters=128

size=1

stride=1

pad=1

activation=leaky

[convolutional]

batch\_normalize=1

filters=256

size=3

stride=1

pad=1

activation=leaky

[shortcut]

from=-3

activation=linear

[convolutional]

batch\_normalize=1

filters=128

size=1

stride=1

pad=1

activation=leaky

[convolutional]

batch\_normalize=1

filters=256

size=3

stride=1

pad=1

activation=leaky

[shortcut]

from=-3

activation=linear

[convolutional]

batch\_normalize=1

filters=128

size=1

stride=1

pad=1

activation=leaky

[convolutional]

batch\_normalize=1

filters=256

size=3

stride=1

pad=1

activation=leaky

[shortcut]

from=-3

activation=linear

[convolutional]

batch\_normalize=1

filters=128

size=1

stride=1

pad=1

activation=leaky

[convolutional]

batch\_normalize=1

filters=256

size=3

stride=1

pad=1

activation=leaky

[shortcut]

from=-3

activation=linear

# Downsample

[convolutional]

batch\_normalize=1

filters=512

size=3

stride=2

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

[shortcut]

from=-3

activation=linear

[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

[shortcut]

from=-3

activation=linear

[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

[shortcut]

from=-3

activation=linear

[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

[shortcut]

from=-3

activation=linear

[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

[shortcut]

from=-3

activation=linear

[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

[shortcut]

from=-3

activation=linear

[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

[shortcut]

from=-3

activation=linear

[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

[shortcut]

from=-3

activation=linear

# Downsample

[convolutional]

batch\_normalize=1

filters=1024

size=3

stride=2

pad=1

activation=leaky

[convolutional]

batch\_normalize=1

filters=512

size=1

stride=1

pad=1

activation=leaky

[convolutional]

batch\_normalize=1

filters=1024

size=3

stride=1

pad=1

activation=leaky

[shortcut]

from=-3

activation=linear

[convolutional]

batch\_normalize=1

filters=512

size=1

stride=1

pad=1

activation=leaky

[convolutional]

batch\_normalize=1

filters=1024

size=3

stride=1

pad=1

activation=leaky

[shortcut]

from=-3

activation=linear

[convolutional]

batch\_normalize=1

filters=512

size=1

stride=1

pad=1

activation=leaky

[convolutional]

batch\_normalize=1

filters=1024

size=3

stride=1

pad=1

activation=leaky

[shortcut]

from=-3

activation=linear

[convolutional]

batch\_normalize=1

filters=512

size=1

stride=1

pad=1

activation=leaky

[convolutional]

batch\_normalize=1

filters=1024

size=3

stride=1

pad=1

activation=leaky

[shortcut]

from=-3

activation=linear

######################

[convolutional]

batch\_normalize=1

filters=512

size=1

stride=1

pad=1

activation=leaky

[convolutional]

batch\_normalize=1

size=3

stride=1

pad=1

filters=1024

activation=leaky

[convolutional]

batch\_normalize=1

filters=512

size=1

stride=1

pad=1

activation=leaky

[convolutional]

batch\_normalize=1

size=3

stride=1

pad=1

filters=1024

activation=leaky

[convolutional]

batch\_normalize=1

filters=512

size=1

stride=1

pad=1

activation=leaky

[convolutional]

batch\_normalize=1

size=3

stride=1

pad=1

filters=1024

activation=leaky

[convolutional]

size=1

stride=1

pad=1

filters=18

activation=linear

[yolo]

mask = 6,7,8

anchors = 10,13, 16,30, 33,23, 30,61, 62,45, 59,119, 116,90, 156,198, 373,326

classes=1

num=9

jitter=.3

ignore\_thresh = .7

truth\_thresh = 1

random=1

[route]

layers = -4

[convolutional]

batch\_normalize=1

filters=256

size=1

stride=1

pad=1

activation=leaky

[upsample]

stride=2

[route]

layers = -1, 61

[convolutional]

batch\_normalize=1

filters=256

size=1

stride=1

pad=1

activation=leaky

[convolutional]

batch\_normalize=1

size=3

stride=1

pad=1

filters=512

activation=leaky

[convolutional]

batch\_normalize=1

filters=256

size=1

stride=1

pad=1

activation=leaky

[convolutional]

batch\_normalize=1

size=3

stride=1

pad=1

filters=512

activation=leaky

[convolutional]

batch\_normalize=1

filters=256

size=1

stride=1

pad=1

activation=leaky

[convolutional]

batch\_normalize=1

size=3

stride=1

pad=1

filters=512

activation=leaky

[convolutional]

size=1

stride=1

pad=1

filters=18

activation=linear

[yolo]

mask = 3,4,5

anchors = 10,13, 16,30, 33,23, 30,61, 62,45, 59,119, 116,90, 156,198, 373,326

classes=1

num=9

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=4

[route]

layers = -1, 11

[convolutional]

batch\_normalize=1

filters=128

size=1

stride=1

pad=1

activation=leaky

[convolutional]

batch\_normalize=1

size=3

stride=1

pad=1

filters=256

activation=leaky

[convolutional]

batch\_normalize=1

filters=128

size=1

stride=1

pad=1

activation=leaky

[convolutional]

batch\_normalize=1

size=3

stride=1

pad=1

filters=256

activation=leaky

[convolutional]

batch\_normalize=1

filters=128

size=1

stride=1

pad=1

activation=leaky

[convolutional]

batch\_normalize=1

size=3

stride=1

pad=1

filters=256

activation=leaky

[convolutional]

size=1

stride=1

pad=1

filters=18

activation=linear

[yolo]

mask = 0,1,2

anchors = 10,13, 16,30, 33,23, 30,61, 62,45, 59,119, 116,90, 156,198, 373,326

classes=1

num=9

jitter=.3

ignore\_thresh = .7

truth\_thresh = 1

random=1

**Lampiran 4**. *Source code counting* larva dengan YOLO V3

import os

from os import walk

import PIL

from PIL import Image

import subprocess

import sys

import datetime

import time

os.system

start = time.time()

img\_dir = "images/"

out\_dir = "output/"

text\_file = open('imgpath.txt', 'w')

"""List of Images Path"""

txt\_img\_list = []

"""Iteration for showing pictures that we want to process"""

for (dirpath, dirnames, filenames) in walk(img\_dir):

txt\_img\_list.extend(filenames)

break

print(txt\_img\_list)

text\_file.write("\n")

"""Process"""

for txt\_name in txt\_img\_list:

"""Save images path"""

text\_file.write('images/%s.jpg\n'%(os.path.splitext(txt\_name)[0]))

text\_file.close()

"""Variable Init"""

d = 1

mytime = 5

total = 0

"""Reading image path"""

with open('imgpath.txt', 'r') as fobj:

for line in fobj:

image\_List = [[num for num in line.split()] for line in fobj]

open("larvaNumber.txt", "w").close()

"""Iteration for detecting images"""

for images in image\_List:

commands = ["darknet.exe", "detector", "test", "data/obj.data", "cfg/small-larva-test.cfg", "newweight/small-larva\_final.weights" ,"-dont\_show", images[0]]

os.system(', '.join(commands))

output = subprocess.check\_output(commands)

output = output.decode("utf-8").split("\n")

"""Count the number of lines that contain "larva"""

numLarva = len([i.split(":")[0] for i in output if i.split(":")[0] == 'larva'])

print("Ada {} larva yang terdeteksi.".format(numLarva))

"""Write total of larva in the frame into larvaNumber.txt"""

with open("larvaNumber.txt", "a") as myfile:

myfile.write("Pada gambar ke {} Ada {} Larva yang terdeteksi\n".format(d, numLarva))

"""Save predictions image into output folder"""

predicted\_image = Image.open("predictions.jpg")

output\_image = "output/predicted\_image%3d.jpg"%d

predicted\_image.save(output\_image)

d+=1

mytime+=5

total=total+numLarva

end = time.time()

print("TOTAL WAKTU PROSES adalah ",end - start, "detik")

print("TOTAL ADA ", total, "LARVA")