BAB 2
LANDASAN TEORI

2.1 Sampah

Sampah digambarkan sebagai bahan, zat, atau benda yang tidak diinginkan, yang mungkin terdiri dari sisa makanan, sisa-sisa, atau sampah. Dari perspektif ekonomi, sampah dapat didefinisikan sebagai bahan sisa yang telah mengalami beberapa tahap proses, baik karena penghapusan bagian penting, pemrosesan, atau hilangnya nilai sosial dan ekonominya sejauh tidak lagi melayani tujuan apa pun [8].

Berdasarkan dengan peraturan yang diuraikan dalam UU No. 18 Tahun 2008 tentang Pengelolaan Sampah, sampah adalah sisa-sisa yang berasal dari tindakan manusia sehari-hari atau kejadian alam yang ada dalam bentuk padat atau semi-padat, terdiri dari zat organik atau anorganik yang berpotensi terurai atau tetap utuh, dan dianggap usang dan akibatnya dibuang [9].

2.1.1 Sampah Organik

Sampah organik merupakan sampah yang berasal dari sisa-sisa makhluk hidup, baik hewan, tanaman, maupun manusia yang dapat terurai secara alamiah di alam (*biodegradable*) [10]. Biasanya sampah jenis ini biasa kita kenal dengan sampah sisa makanan, potongan buah dan sayur, sampah dedaunan, pepohonan, dan rumput-rumputan.

Sampah organik bisa dibedakan lagi secara lebih mendetail ke dalam dua jenis, yaitu sampah organik kering dan sampah organik basah. Sampah organik kering punya kandungan air yang lebih sedikit dibandingkan sampah organik basah. Oleh karena itu, biasanya sampah organik basah akan lebih cepat membusuk sehingga hancur lebih dulu.

2.1.2 Sampah Anorganik

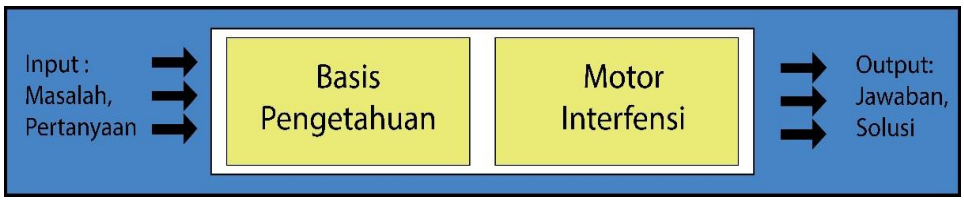
Berbeda dari sampah organik, sampah anorganik tidak dapat terurai secara alami (*undegradable*) karena materialnya tidak berasal dari alam melainkan hasil olahan dari bahan sintetik tertentu [11].

Beberapa contoh sampah anorganik yang sering dijumpai sehari-hari misalnya seperti kantong plastik, kaleng, aluminium, botol kaca, styrofoam, karton, tekstil dan masih banyak lagi. Barang-barang dengan material tersebut tidak dapat membusuk dengan bantuan alam, untuk itu harus diolah kembali oleh manusia atau mesin agar bisa dimanfaatkan menjadi produk baru.

2.2 Artificial Intelligence

Artificial Intelligence atau kecerdasan buatan adalah suatu ilmu yang mempelajari bagaimana komputer dapat digunakan untuk mensimulasikan dan memperluas fungsi kinerja seperti otak manusia. Sehingga komputer dapat memiliki kemampuan layaknya manusia seperti belajar, penyimpangan, menyelesaikan masalah, memori, pengetahuan dan penambahan bahasa alami manusia [12].

Pembuatan suatu aplikasi kecerdasan buatan terdapat dua bagian utama yang sangat dibutuhkan, diantaranya adalah Basis Pengetahuan (*Knowledge Base*) dan Motor Inferensi (*Inference Engine*). Pada basis pengetahuan yakni bersifat fakta-fakta, teori, pemikiran dan hubungan satu dengan yang lainnya. Sedangkan untuk Motor Inferensi merupakan kemampuan untuk menarik kesimpulan berdasarkan pengetahuan dan pengalaman. Penerapan konsep *artificial intelligence* pada komputer dijelaskan pada Gambar 2.1.



Gambar 2. 1 Penerapan Konsep *Artificial Intelligence*

2.3 Machine Learning

Machine learning atau pembelajaran mesin adalah suatu pendekatan dalam *artificial intelligence* yang sedang marak digunakan untuk menirukan perilaku manusia guna menyelesaikan suatu masalah atau melakukan automasi. Setidaknya terdapat dua aplikasi utama dalam *machine learning* yakni klasifikasi dan prediksi. *Machine learning* memiliki ciri khas diantaranya adalah proses pelatihan, pembelajaran atau *training* [13].

Terdapat beberapa tipe algoritma yang digunakan pada *machine learning* diantaranya adalah *supervised learning*, *unsupervised learning*, *reinforcement learning*, dan *evolutionary learning*. Tipe yang sering digunakan banyak orang dalam *machine learning* ialah *supervised learning*. *Supervised learning* adalah suatu *training set* dari banyak contoh dengan respon (target) yang benar disediakan. Berdasarkan *training set* ini, algoritma digeneralisasi untuk merespon dengan benar semua input yang tepat.

Dalam pembelajaran *machine learning*, terdapat tiga kategori utama yaitu:

a. *Supervised Learning*

   Pada *supervised learning*, data yang dimiliki dilengkapi dengan label/kelas yang menunjukkan klasifikasi atau kelompok data tersebut berada. Model yang dihasilkan adalah model prediksi dari data yang telah diberi label.

b. *Unsupervised Learning*

   Pada *unsupervised learning*, data pembelajaran tidak memiliki label/kelas sehingga harus mencari struktur dari data yang ada, kemudian melakukan pengelompokan berdasarkan informasi yang dimiliki.

c. *Reinforcement Learning*

   Pada *reinforcement learning*, pembelajaran terhadap apa yang akan dilakukan (bagaimana memetakan situasi ke dalam aksi) untuk mendapatkan *reward* yang maksimal. Pembelajar tidak diberitahu aksi mana yang akan diambil, tetapi lebih kepada menemukan aksi mana yang dapat memberikan *reward* maksimal dengan mencoba menjalankannya.

2.4 Computer Vision

Computer vision merupakan salah satu cabang ilmu pengetahuan yang bertujuan untuk membuat suatu keputusan yang berguna untuk mengenali objek fisik nyata dan keadaan berdasarkan sebuah gambar atau citra [14]. *Computer vision* menjadikan komputer "*acts like human sight*", sehingga mendekati kemampuan manusia dalam menangkap informasi visual. Kemampuan itu diantaranya adalah:

a. *Object detection*: Mengenali sebuah objek ada pada scene dan mengetahui dimana batasanya.
b. *Recognition*: Menempatkan label pada objek.
c. *Description*: Menugaskan properti kepada objek.
d. *3D Inference*: Menafsirkan adegan 3D dari 2D yang dilihat.
e. *Interpreting motion*: Menafsirkan gerakan.

2.5 Deteksi Objek

Deteksi objek merupakan teknik yang berkaitan dengan mendeteksi suatu objek dari *class* tertentu (seperti manusia, bangunan, atau mobil) dalam gambar atau video digital [15]. Mesin akan memproses gambar atau video dan akan menghasilkan keluaran setiap objek yang terdeteksi beserta label yang sesuai dengan *class*. Mesin juga akan menampilkan *bounding box* (kotak pembatas) pada objek yang terdeteksi dan akan menampilkan nilai akurasi dari pendeteksian tersebut. Metode deteksi objek dikategorikan menjadi tiga kelas utama, yaitu deteksi berdasarkan gerakan, deteksi berdasarkan fitur penampilan, dan deteksi berbasis *convolutional neural network* (CNN).

Deteksi berdasarkan fitur penampilan memfokuskan pada fitur atau rupa bentuk dari objek, seperti warna, bentuk, arah tampilan objek dan sebagainya. Maka dari itu pemilihan fitur sangat berpengaruh besar pada keseluruhan kinerja sistem dalam melakukan deteksi.

Mendeteksi objek bergerak artinya mengenali gerakan fisik suatu objek di tempat atau wilayah tertentu. Cara kerjanya dengan melakukan segmentasi diantara objek yang bergerak dengan area yang tidak berubah. Objek yang bergerak tersebut

dapat dilacak dan dianalisis gerakannya dengan cara mencari objek target yang bergerak dari setiap frame pada video atau hanya pada saat objek menunjukkan pergerakan pertamanya pada video [16].

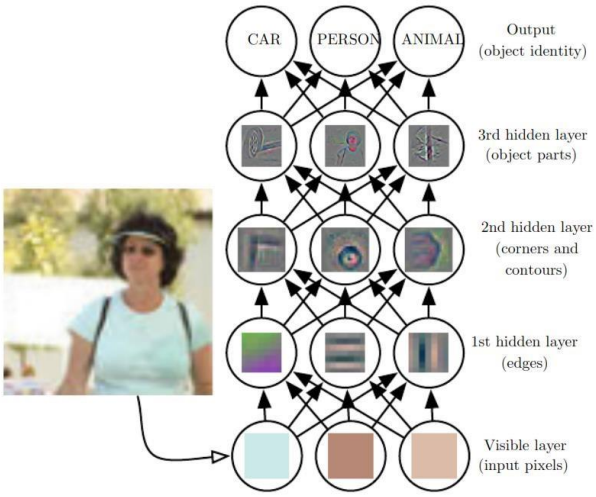
Sedangkan pada deteksi berbasis *Convolutional Neural Network* (CNN) yang merupakan sebuah *neural network* yang terdiri dari beberapa lapisan berbeda seperti lapisan *input layer* (masukan), setidaknya ada satu *hidden layer* (lapisan tersembunyi), dan *output layer* (lapisan keluaran) . *Hidden layer* bertugas seperti filter yang menerima masukan awal, kemudian merubahnya menggunakan fitur tertentu, lalu memindahkannya ke lapisan berikutnya. Perubahan yang dilakukan pada setiap *hidden layer* berbeda - beda, misalkan satu *layer* mengidentifikasi objek berwarna hijau, kemudian *hidden layer* berikutnya mungkin menyimpulkan bahwa objek itu berupa daun. Kesimpulannya, semakin banyak *layer* yang ada, objek yang dideteksi bisa beragam.

Deteksi objek atau biasa disebut dengan *object detection* adalah suatu proses yang digunakan untuk menentukan keberadaan objek tertentu dalam suatu citra digital. Proses deteksi dapat dilakukan dengan berbagai macam metode yang pada dasarnya melakukan pembacaan fitur-fitur dari seluruh objek pada citra *input*. Fitur dari objek pada citra akan dibandingkan dengan fitur dari objek referensi atau *template* lalu dibandingkan dan ditentukan apakah objek yang dideteksi termasuk objek yang ingin dideteksi atau tidak.

2.6 Deep Learning

Deep learning adalah suatu cabang ilmu dari *machine learning* yang terinspirasi dari kortek manusia dengan mengimplementasikan jaringan syaraf buatan yang banyak memiliki *hidden layer* [17]. Menurut Abu Ahmad, *deep learning* adalah teknik yang terdapat pada *neural network* dengan menggunakan teknik tertentu yakni seperti *Restricted Boltzmann Machine* (RBM) untuk meningkatkan kecepatan proses pembelajaran pada *neural network* yang menggunakan *layer* yang banyak atau lebih dari tujuh *layer* [18]. Beberapa jenis *deep learning* yakni *Deep Auto Encoder* (DAE), *Deep Belief Nets* (DBN), *Convolutional Neural Network* (CNN),

dan lain lain. Ilustrasi dari proses *deep learning* dari *input* hingga mendapatkan *output* dapat dilihat pada Gambar 2.2.



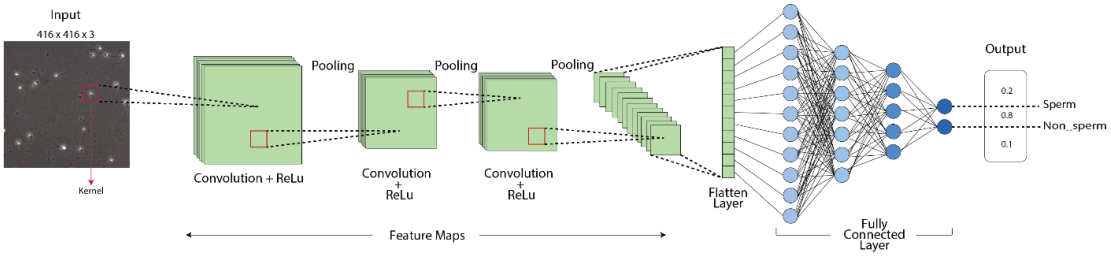
Gambar 2. 2 Ilustrasi dari model *deep learning*

Sulit bagi komputer untuk memahami arti data *input* sensorik mentah seperti gambar diatas yang dipresentasikan dengan sekumpulan *pixel*. *Deep learning* mampu menyelesaikan kesulitan tersebut dengan memecah pemetaan rumit yang diinginkan menjadi serangkaian pemetaan sederhana bersarang. Dari gambar diatas terdapat lima proses *deep learning* dalam menyelesaikan permasalahan tersebut. Pertama *layer* terlihat yang masih berupa *pixel* warna dan kemudian akan diekstrak menjadi bentuk yang abstrak *layer* ini disebut dengan *hidden layer*. Dalam *hidden layer* ini tahap pertama mampu mengidentifikasi garis atau tepi, *hidden layer* kedua mampu mengidentifikasi pencarian sudut dengan mudah serta kontur diperpanjang atau kumpulan tepi, berikutnya *hidden layer* ketiga sudah mampu mendeteksi seluruh bagian dari objek tertentu. Akhirnya, deskripsi gambar dalam hal bagian-bagian objek yang telah dikelola dapat digunakan untuk mengenali hasil objek yang ada dalam gambar [19].

2.7 Convolutional Neural Network

Convolutional Neural Network (CNN) merupakan salah satu cabang algoritma *deep learning*, digunakan pada *computer vision* untuk menyelesaikan suatu kasus atau masalah tertentu seperti mengklasifikasi dan mendeteksi suatu objek pada

gambar atau video. CNN termasuk kedalam variasi *Multi Layer Perceptron* (MLP) yang terinspirasi dari jaringan syaraf pada manusia. Ciri pada CNN yakni memiliki susunan neuron 3D (tinggi, lebar, dan kedalaman). Lebar dan tinggi merupakan suatu ukuran pada *layer* sedangkan untuk kedalaman yakni menunjukkan pada jumlah *layer* [20]. Gambar 2.3 merupakan ilustrasi arsitektur dari *Convolutional Neural Network* (CNN).

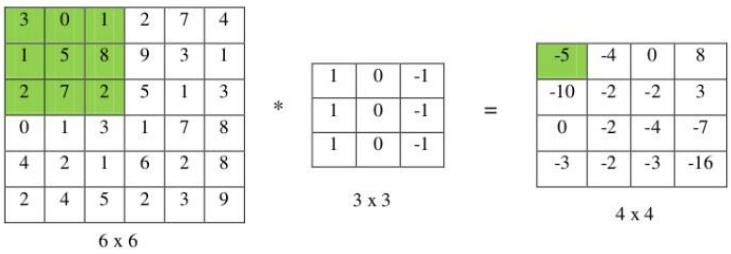


Gambar 2. 3 Arsitektur *Convolutional Neural Network* (CNN)

Gambar 2.3 menggambarkan secara sederhana arsitektur umum dari sebuah *Convolutional Neural Network* (CNN) yang dirancang untuk melakukan klasifikasi gambar.

2.7.1 Konvolusi

Konvolusi merupakan suatu proses memanipulasi citra dengan menggunakan eksternal *mask* atau *subwindows* untuk menghasilkan suatu citra baru. Definisi konvolusi secara matematis ialah jumlah total dari perkalian antara elemen yang bersesuaian (memiliki koordinat yang sama) dalam dua matriks atau dua vektor [21]. Contoh gambaran konvolusi diilustrasikan pada Gambar 2.4.



Gambar 2. 4 Ilustrasi Konvolusi Matriks 6 x 6

Gambar 2.4 menampilkan proses untuk konvolusi pada CNN, langkah pertama dapat di lihat pada matriks *input* 6 × 6 yang akan dioperasikan dengan kernel / filter 3 x 3. Proses pengoperasian ini nilai matriks *input* akan dikalikan dengan nilai matriks filter mulai dari bagian yang telah dihijaukan dan seterusnya dengan langkah / *stride* 1. Sehingga nilai yang dihasilkan pada setiap perkalian dengan filter akan mengisi nilai matriks baru. Berdasarkan contoh pada Gambar 2.4 hasil *ouput* matriks yang dihasilkan ialah matriks 4 x 4 disebabkan oleh penggunaan langkah / *stride* 1. Citra *output* akan menghasilkan matriks yang berbeda dengan citra *input*, hasil tersebut di dapat dari persamaan 1 berikut

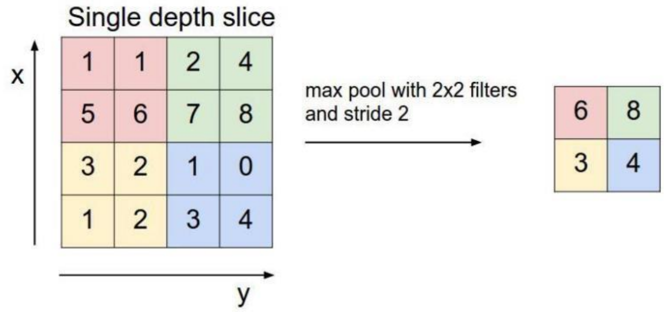
$Hasil = Input - Kernel + 1$ ……………………………………………….(1)

$Hasil = 6 - 3 + 1 = 4$

Maka akan menghasilkan matriks baru yaitu 4 x 4.

2.7.2 Pooling

Pooling berfungsi untuk mereduksi ukuran spasial dari citra dan jumlah parameter dalam jaringan. *Pooling* juga dapat mempercepat komputasi dan mengontrol terjadinya *overfitting* [22]. Pada Gambar 2.5 merupakan ilustrasi dari proses *pooling*.



Gambar 2. 5 Ilustrasi *matrix pooling* menggunakan 2 x 2 *filter* dan 2 *stride*

Pada Gambar 2.5 menampilkan contoh dari *pooling* yakni jenis *max pooling*. *Max pooling* ini mengambil nilai terbesar dari bagian tersebut, pada contoh Gambar 2.5 menggunakan filter 2 x 2 sehingga *max pooling* akan mengambil nilai terbesar dari setiap bagian. Dapat di lihat pada bagian warna merah muda terdapat nilai baris

dan kolom diantaranya 1, 1, 5, dan 6. Sehingga dari bagian warna merah muda tersebut diambil angka yang terbesar yakni 6 untuk dijadikan nilai pada matriks baru.

2.7.3 Fully Connected Layer

Fully Connected Layer adalah suatu lapisan yang terhubung secara penuh atau keseluruhan, lapisan *neuron* ini terhubung langsung dengan neuron lainnya dengan dua lapisan yang berdekatan tanpa terhubung dengan lapisan apapun [23]. *Fully Connected Layer* merupakan suatu *layer* yang dipakai dalam melakukan transformasi pada suatu dimensi data sehingga dapat diklasifikasikan secara linier [24].

Output berupa *activation map* merupakan hasil yang masih berupa dimensi lebar (*width*), tinggi (*height*) dan *channel*. Maka dilakukan perubahan (*flatten*) agar *output* menjadi vektor sehingga data dapat diklasifikasikan secara linear. *Output* diubah menjadi 1 x 1 dimensi dimana jumlahnya didapat dari perkalian antara lebar (*width*), tinggi (*height*) dan *channel* ditambah jumlah bias.

2.8 YOLO (*You Only Look Once*)

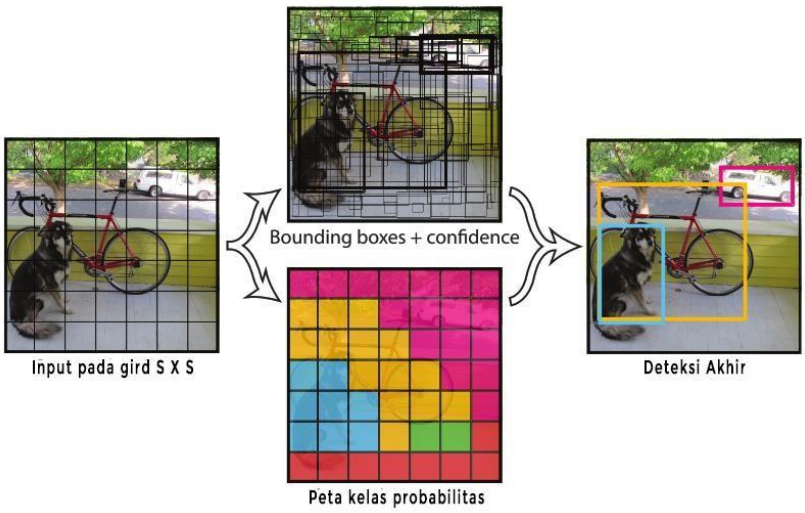
You Only Look Once atau YOLO merupakan suatu pendekatan terbaru untuk deteksi objek**.** YOLO berbeda dengan penelitian sebelumnya yang mendeteksi suatu objek menggunakan kembali pengklasifikasinya, melainkan YOLO membingkai deteksi objek sebagai masalah regresi dengan *bounding boxes* (kotak pembatas) yang terpisah secara spasial dan probabilitas kelas yang terkait. Proses gambar di dalam YOLO ini sederhana dan mudah dapat dilihat pada Gambar 2.6.

Gambar 2. 6 Proses Sistem Deteksi YOLO

Pada Gambar 2.6 tahapan awal dalam proses gambar dalam YOLO yakni pertama mengubah ukuran gambar *input* menjadi 448 x 448, selanjutnya menjalankan jaringan konvolusi tunggal pada gambar, dan yang terakhir yakni membatasi deteksi yang dihasikan dengan *model's confidence*. YOLO membagi gambar input menjadi *grid* $S \times S$. Jika pusat dari suatu objek jatuh kedalam sel *grid*, maka sel *grid* tersebut yang akan bertanggung jawab untuk mendeteksi objek tersebut. Setiap sel *grid* memprediksi B *bounding boxes* dan *confidence scores*. *Confidence score* mempresentasikan seberapa yakin model bahwa *bounding boxes* itu berisi suatu objek dan seberapa akurat objek tersebut diprediksi. Umumnya *confidence* didefinisikan sebagai $Pr(Object) * IOU_{pred}^{truth}$, jika tidak ada objek pada sel tersebut maka nilai *confidence* adalah nol. Setiap sel *grid* juga memprediksi probabilitas kelas bersyarat C atau $Pr(Class_i|Object)$. Objek yang diprediksi hanya satu set probabilitas kelas per sel *grid*, terlepas dari jumlah *boxes B*. Dalam pengujian mengalikan probabilitas kelas bersyarat dan prediksi *individual box confidence*,

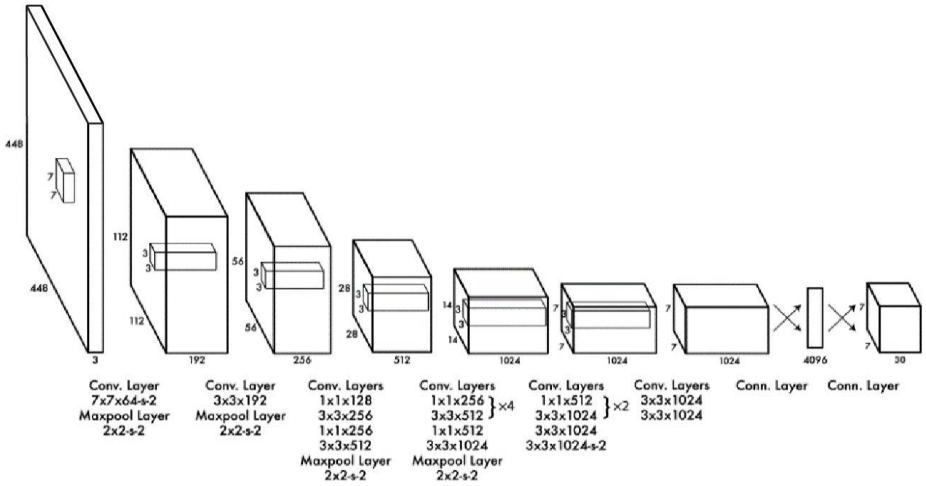
$$Pr(Class_i|Object) * Pr(Object) * IOU_{pred}^{truth} = Pr\,(Class_i) * IOU_{pred}^{truth} \quad \ldots\ldots(2)$$

Persamaan 2 merupakan skor *confidence* kelas khusus untuk setiap kotak. Skor ini mengkodekan probabilitas kelas itu muncul didalam kotak dan sebar baik kotak yang diprediksi cocok dengan objek. Gambar 2.7 merupakan gambaran dari prediksi *bounding boxes B* yang dikodekan sebagai tensor $S \times S \times (B * 5 + C)$ [25].



Gambar 2. 7 Model Deteksi menggunakan *Bounding Box*

Pada Gambar 2.7 YOLO diimplementasikan sebagai *convolutional neural network.* Arsitektur ini terinspirasi oleh model GoogleNet untuk klasifikasi gambar. Jaringan pada YOLO ini memiliki 24 lapisan konvolusi diikuti oleh 2 *fully connected layers.* Secara sederhana menggunakan 1 × 1 reduksi *layer* yang diikuti oleh 3 × 3 *convolutional layers*. Prediksi dari *output* akhir jaringan YOLO ini adalah tensor 7 × 7 × 30. Untuk gambaran arsitektur jaringannya dapat di lihat pada Gambar 2.8 [26].

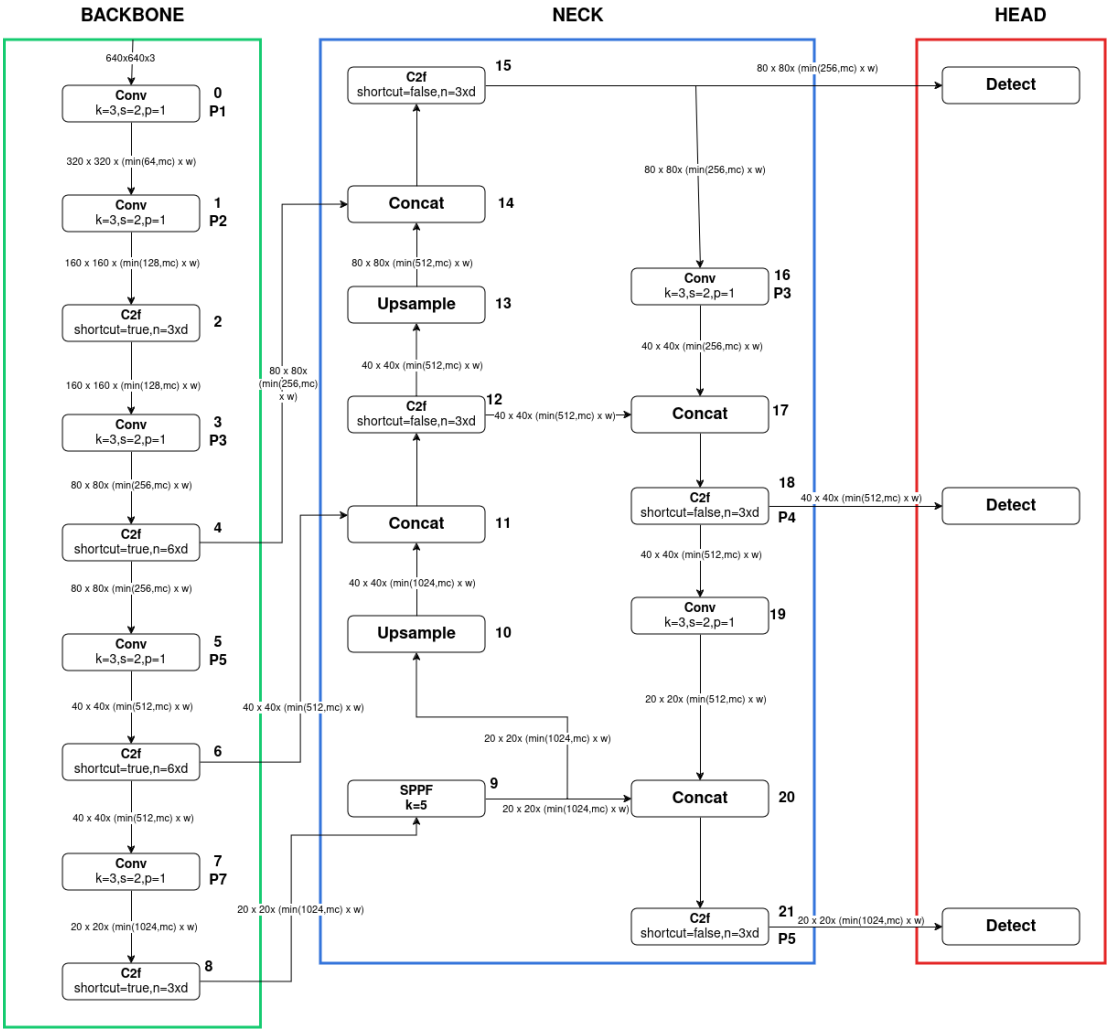


Gambar 2. 8 Arsitektur YOLO

Arsitektur YOLO yang ditunjukkan pada Gambar 2.8 merupakan salah satu varian dari YOLO yang populer. Arsitektur ini telah terbukti efektif untuk tugas deteksi objek *real-time*.

2.9 *You Only Look Once* V8 (YOLO V8)

YOLO v8 adalah versi terbaru dari algoritma YOLO dan dirilis pada tahun 2020. Ini menampilkan beberapa peningkatan dibandingkan versi sebelumnya, termasuk jaringan tulang punggung yang lebih efisien, prediksi multi-skala, dan sistem jangkar baru. Arsitektur YOLO v8 terdiri dari jaringan tulang punggung (*backbone network*), leher (*neck*), dan kepala (*head*). Jaringan tulang punggung menggunakan *Feature Pyramid Network* (FPN) untuk mengekstraksi fitur dari gambar *input*, sedangkan leher menggunakan serangkaian *Cross-Layer Connection* (CLC) untuk menyempurnakan fitur ini. Kepala mengambil fitur yang disempurnakan dan memprediksi kotak pembatas, skor kelas objek, dan akurasi untuk setiap objek dalam citra. Memiliki 105 layer [27].



Gambar 2. 9 Arsitektur YOLO v8

Gambar 2.9 menunjukkan arsitektur YOLO v8 terdiri dari tiga bagian utama, yaitu *Backbone* adalah arsitektur pembelajaran mendalam yang bertindak sebagai ekstraktor fitur dari gambar yang dimasukkan, *Neck* menggabungkan fitur yang diperoleh dari berbagai lapisan modul *Backbone*, dan *Head* memprediksi kelas dan kotak pembatas objek yang merupakan keluaran akhir yang dihasilkan oleh model deteksi objek. Berikut adalah penjelasan rinci tentang alur blok proses dalam Arsitektur YOLO v8.

A. Blok 1-5: *Backbone* (*Feature Pyramid Network* - FPN)

Gambar *input* diumpankan ke blok *Convolution* untuk mengekstraksi fitur. Fitur dari blok *convolution* diumpankan ke blok *downsample* untuk

menurunkan resolusi gambar dari 640 x 640 menjadi 320 x 320 piksel. Fitur dari blok *downsample* diumpankan ke blok *bottleneck* untuk mengekstraksi fitur lebih lanjut. Fitur dari blok *bottleneck* diumpankan ke blok CSP *Route* untuk menggabungkan fitur dari dua jalur yang berbeda.

Hasil dari blok CSP *Route* disimpan sebagai *output* dari *stage* 1 FPN.

Langkah 1-5 diulang untuk *stage* 2-5 FPN, dengan *input* gambar dan resolusi yang berbeda:

1. *Stage* 2: *Input* 320 x 320 piksel, *output* 160 x 160 piksel
2. *Stage* 3: *Input* 160 x 160 piksel, *output* 80 x 80 piksel
3. *Stage* 4: *Input* 80 x 80 piksel, *output* 40 x 40 piksel
4. *Stage* 5: *Input* 40 x 40 piksel, *output* 20 x 20 piksel

Tujuannya untuk mengekstraksi fitur dari gambar *input* pada berbagai skala untuk menghasilkan representasi yang lebih kaya.

B. Blok 6-10: *Neck* (*Cross-Layer Connection* – CLC)

Fitur dari *stage* 1 FPN diumpankan ke blok *Conv* untuk mengekstraksi fitur. Fitur dari blok *Convolution* diumpankan ke blok *upsample* untuk meningkatkan resolusi gambar menjadi 320 x 320 piksel. Fitur yang di-*upsample* digabungkan dengan fitur dari *stage* 2 FPN.

Langkah 2-3 diulang untuk menggabungkan fitur dari *stage* 1-5 FPN:

1. *Stage* 2: *upsample* ke 160 x 160 piksel, gabungkan dengan *stage* 3 FPN
2. *Stage* 3: *upsample* ke 80 x 80 piksel, gabungkan dengan *stage* 4 FPN
3. *Stage* 4: *upsample* ke 40 x 40 piksel, gabungkan dengan *stage* 5 FPN

Tujuannya untuk meningkatkan aliran informasi dalam arsitektur YOLO v8 dengan menghubungkan fitur dari berbagai skala gambar.

C. Blok 11-15: *Head* 1 (CSP - *Cross Stage Partial*)

Fitur dari *stage* 3 FPN diumpankan ke blok *Convolution* untuk mengekstraksi fitur. Fitur dari blok *Convolution* diumpankan ke blok *bottleneck* untuk mengekstraksi fitur lebih lanjut. Fitur dari blok *bottleneck* diumpankan ke blok CSP *Route* untuk menggabungkan fitur dari dua jalur yang berbeda.

Hasil dari blok CSP *Route* diumpankan ke blok *Convolution* untuk mengekstraksi fitur akhir. Langkah 1-4 diulang untuk fitur dari *stage* 4 dan 5 FPN.

Hasil dari ketiga *head* (stage 3, 4, dan 5) digabungkan.

Tujuannya untuk mendeteksi objek pada skala kecil, menengah, dan besar.

D. Blok 16-20: *Head* 2 (CSP - *Cross Stage Partial*)

Mirip dengan *head* 1, tetapi hanya menggunakan fitur dari *stage* 2 dan 3 FPN.

Tujuannya untuk mendeteksi objek pada skala menengah dan besar.

E. Blok 21: *Detect Block*

Hasil gabungan diumpankan ke blok *convolution* untuk mengekstraksi fitur akhir.

Fitur dari blok *convolution* diumpankan ke blok *detection* untuk mendeteksi objek dalam gambar. Hasil deteksi objek ditampilkan.

Tujuannya untuk mendeteksi objek dalam gambar dengan berbagai ukuran, kelas, dan lokasi.

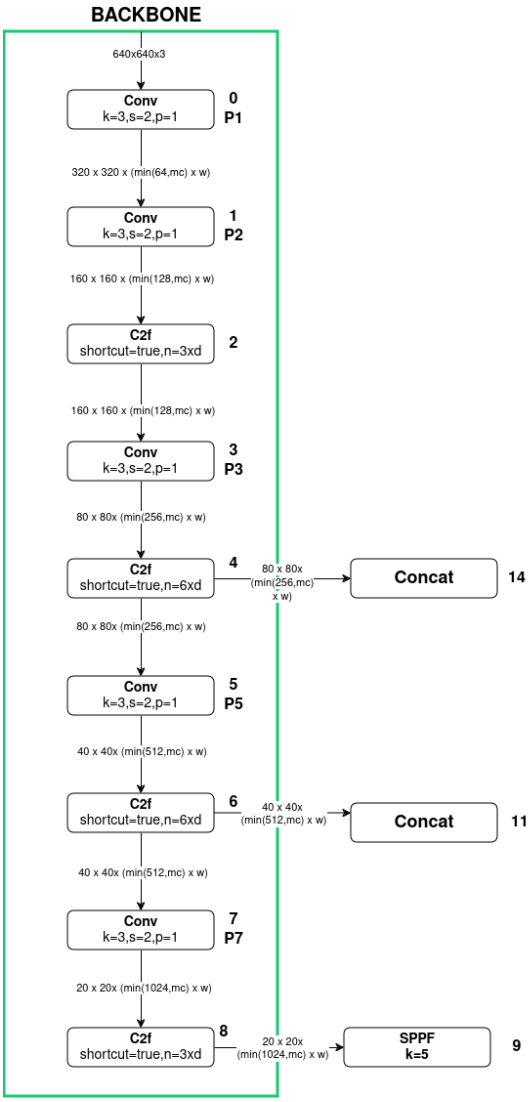
2.9.1. Backbone

Backbone YOLO v8 menggunakan FPN dengan lima *stage*. *Stage* 1 menggunakan gambar *input* dengan resolusi 640 x 640 piksel. *Stage* 2 menggunakan gambar *input* dengan resolusi 320 x 320 piksel. *Stage* 3 menggunakan gambar *input* dengan resolusi 160 x 160 piksel. *Stage* 4 menggunakan gambar *input* dengan resolusi 80 x 80 piksel. *Stage* 5 menggunakan gambar *input* dengan resolusi 40 x 40 piksel.

Setiap stage dalam FPN terdiri dari dua blok, yaitu *Convolution* dan *Upsample*. *Convolution block* menggunakan *convolutional layer* untuk mengekstraksi fitur dari gambar *input*. Blok *upsample* menggunakan *bilinear interpolation* untuk meningkatkan resolusi gambar *input*.

Fitur dari *stage* yang berbeda dalam FPN digabungkan untuk menghasilkan representasi yang lebih kaya. Fitur dari *stage* yang lebih tinggi memiliki skala yang lebih besar dan menangkap informasi semantik yang lebih global. Fitur dari *stage*

yang lebih rendah memiliki skala yang lebih kecil dan menangkap informasi detail yang lebih lokal.



Gambar 2. 10 Arsitektur *Backbone*

Pada Gambar 2.10 arsitektur YOLO (*You Only Look Once*) versi 8, bagian *backbone* adalah komponen utama yang bertanggung jawab untuk ekstraksi fitur dari citra *input*. *Backbone* ini terdiri dari serangkaian lapisan konvolusional yang bertujuan untuk mengidentifikasi dan mengekstrak fitur penting dari citra pada berbagai tingkat representasi. *Backbone* pada YOLO v8 adalah pondasi yang memungkinkan jaringan untuk memahami konten visual dari citra *input*. *Backbone* pada YOLO v8 biasanya terdiri dari kombinasi beberapa blok konvolusional dan

teknik-teknik pengolahan fitur untuk menghasilkan representasi yang kaya dan informatif dari citra *input*.

$$Output\ Size = \left(\frac{Input\ size - kernel\ size + 2\ x\ padding}{Stride}\right) + 1 \quad \text{.....(3)}$$

Ukuran gambar pada arsitektur YOLO v8 dibagi menjadi beberapa bagian dengan ukuran berbeda (misalnya, 80 x 80, 40 x 40, dan 20 x 20) sebagai bagian dari strategi *multi-scale feature extraction*. Tujuannya adalah untuk menangkap informasi pada berbagai tingkat resolusi yang dapat meningkatkan akurasi deteksi objek pada berbagai ukuran dan skala.

1. *Multi-Scale Feature Extraction*

   Menggunakan gambar dengan berbagai ukuran memungkinkan model untuk menangkap fitur pada berbagai skala. Ini penting karena objek dalam gambar bisa muncul dalam berbagai ukuran, dari yang sangat kecil hingga sangat besar. Dengan mengekstrak fitur dari gambar yang diperkecil, model dapat mendeteksi objek kecil dengan lebih baik, sementara fitur dari gambar yang lebih besar dapat membantu dalam mendeteksi objek yang lebih besar.
2. *Handling Objects of Different Sizes*
   a. Ukuran gambar kecil (misalnya, 20 x 20):

      Membantu dalam mendeteksi objek yang sangat kecil yang mungkin tidak terlihat pada resolusi yang lebih besar. Informasi detail dari objek kecil bisa lebih mudah diidentifikasi pada tingkat ini.

   b. Ukuran gambar sedang (misalnya, 60 x 60):

      Baik untuk mendeteksi objek dengan ukuran menengah yang mungkin tidak terlalu kecil atau terlalu besar. Ini mencakup objek yang sering ditemukan dalam banyak aplikasi deteksi objek.

   c. Ukuran gambar besar (misalnya, 80 x 80):

      Membantu dalam mendeteksi objek yang lebih besar dengan mengurangi kompleksitas komputasi dibandingkan jika hanya menggunakan ukuran gambar asli yang sangat besar. Ini

memungkinkan model untuk mengenali pola yang lebih luas dan konteks sekitar objek besar.

3. *Feature Pyramid Networks* (FPN)

   Bagian *neck* dari YOLO v8 sering menggunakan konsep *Feature Pyramid Networks* (FPN), yang menggabungkan fitur dari berbagai skala untuk membuat prediksi yang lebih akurat. Dengan menggabungkan fitur dari berbagai ukuran gambar, model dapat memanfaatkan informasi detail dari berbagai tingkat resolusi untuk membuat keputusan deteksi yang lebih baik.

4. *Improving Robustness and Accuracy*

   Menggunakan fitur dari berbagai ukuran gambar meningkatkan robusta model terhadap variasi skala dan resolusi objek. Ini memastikan bahwa model tidak hanya bergantung pada satu tingkat resolusi, tetapi juga mampu mengenali objek dalam berbagai kondisi dan perspektif.

5. *Reducing Computational Load*

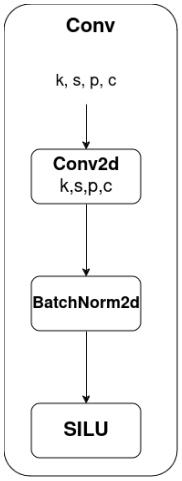
   Dengan memproses gambar pada berbagai ukuran yang lebih kecil, model dapat mengurangi beban komputasi dibandingkan jika harus memproses gambar penuh resolusi sepanjang waktu. Ini membantu dalam membuat deteksi objek lebih efisien dan cepat, terutama penting untuk aplikasi *real-time*.

Pembagian ukuran gambar menjadi beberapa bagian (80 x 80, 60 x 60, 20 x 20) dalam arsitektur YOLO v8 bertujuan untuk meningkatkan kemampuan model dalam menangkap fitur dari berbagai skala, menangani objek dari berbagai ukuran, dan menggabungkan informasi dari berbagai tingkat resolusi untuk meningkatkan akurasi deteksi. Strategi ini juga membantu dalam mengurangi beban komputasi dan meningkatkan efisiensi proses deteksi objek.

A. *Convolutional Block* (Conv Block)

Convolutional block layer adalah kumpulan lapisan konvolusional yang dikombinasikan untuk memproses data *input* melalui beberapa tahapan. Proses konvolusi memanfaatkan apa yang disebut sebagai filter. Seperti layaknya

gambar, filter memiliki ukuran tinggi, lebar, dan tebal tertentu. Filter ini diinisialisasi dengan nilai tertentu (*random* atau menggunakan teknik tertentu seperti Glorot), dan nilai dari filter inilah yang menjadi parameter yang akan di-*update* dalam proses *learning*.



Gambar 2. 11 Arsitektur *Convolutional Block*

Gambar 2.11 adalah blok paling dasar dalam arsitektur yang terdiri dari lapisan *Conv2d*, lapisan *BatchNorm2d*, dan fungsi aktivasi SiLU.

1. *Conv2dLayer*

   Konvolusi adalah operasi matematika yang melibatkan menggeser matriks kecil (disebut kernel atau filter) di atas data masukan, melakukan perkalian berdasarkan elemen, dan menjumlahkan hasilnya untuk menghasilkan peta fitur. “2D” di *Conv2D* mengacu pada fakta bahwa konvolusi diterapkan dalam dua dimensi spasial, biasanya tinggi dan lebar.

   a. K

      Jumlah filter atau kernel mewakili kedalaman volume keluaran, dan setiap filter bertanggung jawab untuk mendeteksi fitur berbeda dalam masukan.

b. S

Stride adalah ukuran langkah di mana filter/kernel meluncur di atas masukan. Langkah yang lebih besar mengurangi dimensi spasial volume keluaran.

c. P

Padding adalah batas tambahan angka nol yang ditambahkan ke masukan di setiap sisi. Ini membantu melestarikan informasi spasial dan dapat digunakan untuk mengontrol dimensi spasial volume keluaran.

d. C

Jumlah *channel* di *input*. Misalnya, dalam gambar RGB, c akan menjadi tiga (satu saluran untuk setiap warna, yaitu merah, hijau, dan biru).

2. *BatchNorm2d Layer*

Batch Normalization (BatchNorm2d) adalah teknik yang digunakan dalam jaringan saraf dalam untuk meningkatkan stabilitas pelatihan dan kecepatan konvergensi. Dalam konteks jaringan saraf konvolusional (CNN), lapisan BatchNorm2d secara khusus menerapkan normalisasi *batch* ke masukan 2D, yang biasanya merupakan keluaran dari lapisan konvolusional. Ini memastikan bahwa nomor yang melewati jaringan tidak terlalu besar atau terlalu kecil. Ini membantu mencegah masalah selama pelatihan.

3. SiLU *Activation Function*

SiLU, singkatan dari *Sigmoid Linear Unit*, adalah fungsi aktivasi yang digunakan dalam jaringan saraf. Ia juga dikenal sebagai fungsi aktivasi *Swish*.

Fungsi aktivasi SiLU didefinisikan pada persamaan 4 berikut:

$SiLU\ (x)\ =\ x\ .\sigma\ (x)$ ……(4)

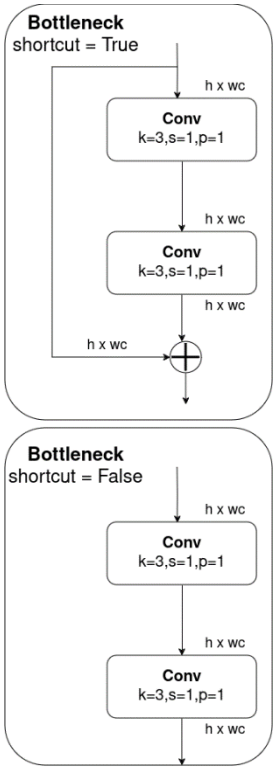
dimana σ(x) adalah fungsi sigmoid, yang diberikan oleh persamaan 5 :

$\sigma\ (x)\ =\ \frac{1}{(1+e^{-x})}$ ……...(5)

Fungsi sigmoid itu sendiri menekan bilangan apa pun antara tak terhingga positif dan negatif ke nilai antara 0 dan 1. Jadi, SiLU pada dasarnya mengambil masukan, menskalakannya dengan nilai antara 0 dan 1 berdasarkan keluaran fungsi sigmoid untuk masukan tersebut, dan mengembalikan nilai hasil. Karakteristik utama SiLU adalah memungkinkan gradien halus, yang dapat bermanfaat selama pelatihan jaringan saraf. Gradien halus dapat membantu menghindari masalah seperti hilangnya gradien, yang dapat menghambat proses pembelajaran di jaringan saraf dalam (*deep neural networks*).

B. Bottleneck Block

Bottleneck block adalah jenis blok residual yang sering digunakan dalam jaringan konvolusional dalam arsitektur modern, seperti ResNet. Tujuannya untuk mengurangi jumlah parameter dan komputasi, sambil mempertahankan performa tinggi dalam ekstraksi fitur.



Gambar 2. 12 Arsitektur *Bottleneck Block*

Arsitektur *Bottleneck Block* yang ditunjukkan pada Gambar 2.12 terdiri dari *convolutional block* dengan koneksi pintasan (*shortcut*). Jika *shortcut* bernilai *True* maka pintasan tersebut diimplementasikan di *bottleneck block*, jika tidak, *input* akan dilewatkan melalui dua *convolutional block* secara seri.

1. *Shortcut Connection*

   Shortcut connection, juga dikenal sebagai *skip connection* atau *residual connection*, adalah *shortcut connection* yang melewati satu atau lebih lapisan dalam jaringan. Hal ini memungkinkan gradien mengalir lebih mudah melalui jaringan selama pelatihan, mengatasi masalah gradien yang hilang dan memudahkan model untuk belajar.

   Dalam konteks spesifik *bottleneck block*, *shortcut connection* memungkinkan model melewati blok konvolusional jika diperlukan. Dengan cara ini, model dapat memilih untuk menggunakan pemetaan identitas yang disediakan oleh pintasan, sehingga memudahkan mempelajari fungsi identitas saat diperlukan. Dimasukkannya *shortcut connection* meningkatkan kemampuan model untuk mempelajari representasi kompleks dan meningkatkan pelatihan CNN mendalam yang mencegah masalah hilangnya gradien.

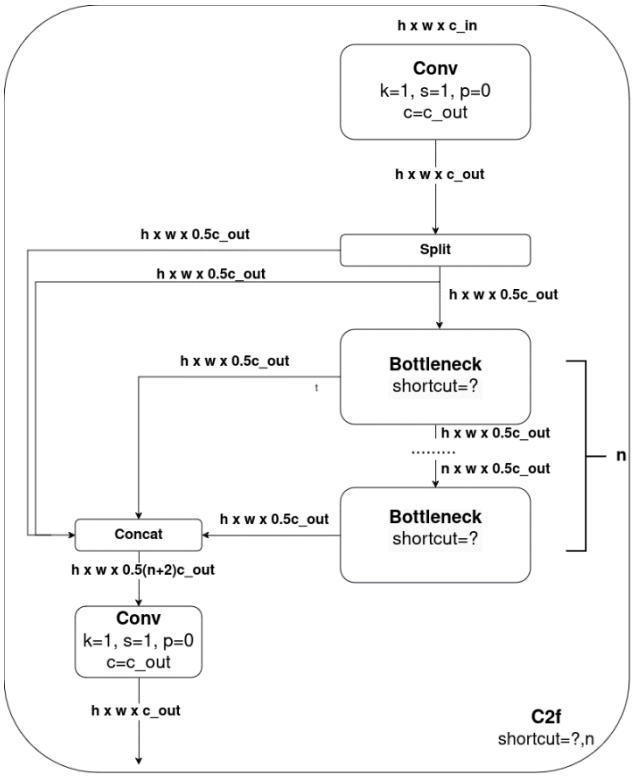
   Masalah gradien hilang merupakan tantangan yang muncul selama pelatihan *deep neural network*, khususnya pada arsitektur dengan banyak lapisan. Hal ini terjadi ketika gradien fungsi kerugian mengenai parameter (bobot) jaringan menjadi sangat kecil karena dipropagasi mundur dari lapisan keluaran ke lapisan masukan selama proses pelatihan.

C. C2f *Block*

C2f *block* merupakan komponen penting yang dirancang untuk meningkatkan efisiensi dan performa jaringan. C2f adalah singkatan dari "*Cross Stage Partial Connections* (CSP) *with 2x Faster*" *block*. Blok ini mengintegrasikan konsep dari *Cross Stage Partial Networks* (CSPNet) dan modifikasi untuk membuat proses lebih cepat, seperti yang diterapkan dalam

YOLO v8. C2f *block* biasanya terdiri dari beberapa lapisan utama dan koneksi parsial yang membantu dalam pembagian dan penggabungan fitur secara efisien.

Untuk menjaga ukuran gambar tetap, *layer* konvolusi dalam blok C2f biasanya menggunakan *padding* yang sesuai. *Padding* ini menambahkan piksel di sekitar gambar *input* agar ketika kernel melintasi gambar, ukuran *output* tetap sama dengan *input*.



Gambar 2. 13 Arsitektur *C2F Block*

Blok C2f yang ditunjukkan pada Gambar 2.13 terdiri dari blok konvolusional yang kemudian peta fitur yang dihasilkan akan dipecah. Satu peta fitur menuju ke blok *Bottleneck* sedangkan peta lainnya langsung menuju ke blok *Concat*. Di blok C2f, jumlah blok *Bottleneck* yang digunakan ditentukan oleh parameter *depth_multiple model*. Pada akhirnya, peta fitur dari blok kemacetan dan peta fitur terpisah digabungkan dan dimasukkan ke dalam blok konvolusional akhir. Setelah matriks melewati proses konvolusi, di mana fitur-

fitur penting dari gambar diekstraksi melalui operasi filter, matriks ini kemudian memasuki tahap C2f (CSPDarknet53 Focus). Tahap C2f ini terdiri dari dua jalur paralel yang mengalir melalui beberapa lapisan konvolusi tambahan dan operasi *pooling*. Salah satu jalur melewati konvolusi lebih dalam, sementara jalur lainnya melewati *shortcut* yang memungkinkan informasi awal gambar dipertahankan. Setelah kedua jalur ini, hasilnya digabungkan kembali untuk memperkaya representasi fitur yang diekstraksi, yang selanjutnya diteruskan ke bagian *neck* dari arsitektur YOLO untuk deteksi lebih lanjut. Proses ini membantu dalam menangkap berbagai aspek dari gambar dan memastikan model memiliki pemahaman yang mendalam tentang fitur penting yang ada dalam gambar.

Proses Convolutional-CSP2 (C2f) berperan penting dalam mengolah fitur yang diekstrak dari gambar input. Proses ini terjadi di berbagai tahap dengan ukuran gambar yang berbeda-beda. Berikut adalah deskripsi rinci mengenai proses C2f dengan berbagai ukuran gambar dan bagaimana mereka dibagi menuju concat, conv, dan SPPF.

1. Ukuan gambar 80 x 80

   Input: Gambar atau fitur dengan ukuran 80 x 80.

   C2f Block:

   Conv2d: Operasi konvolusi pada fitur *input*.

   Feature Split: Fitur yang dihasilkan dibagi menjadi dua bagian:

   Bagian 1: Melalui proses konvolusi berikutnya.

   Bagian 2: Melalui proses *concat* dengan fitur dari jalur lain.

   Tujuan :

   Conv Next: Fitur dari bagian 1 diteruskan ke lapisan konvolusi berikutnya untuk pengolahan lebih lanjut.

Concat: Fitur dari bagian 2 digabungkan (*concatenate*) dengan fitur dari jalur lain untuk memperkaya informasi fitur.

2. Ukuan gambar 40 x 40

   Input: Gambar atau fitur dengan ukuran 40 x 40.

   C2f Block:

   Conv2d: Operasi convolusi pada fitur *input*.

   Feature Split: Fitur yang dihasilkan dibagi menjadi dua bagian:

   Bagian 1: Melalui proses konvolusi berikutnya.

   Bagian 2: Melalui proses *concat* dengan fitur dari jalur lain.

   Tujuan :

   Conv Next: Fitur dari bagian 1 diteruskan ke lapisan konvolusi berikutnya untuk pengolahan lebih lanjut.

   Concat: Fitur dari bagian 2 digabungkan (*concatenate*) dengan fitur dari jalur lain untuk memperkaya informasi fitur.

3. Ukuan gambar 20 x 20

   Input: Gambar atau fitur dengan ukuran 20 x 20.

   C2f Block:

   Conv2d: Operasi konvolusi pada fitur input.

   Feature Split: Fitur yang dihasilkan dibagi menjadi dua bagian:

   Bagian 1: Langsung menuju ke *Spatial Pyramid Pooling - Fast* (SPPF).

   Bagian 2: Melalui proses *concat* dengan fitur dari jalur lain.

Tujuan :

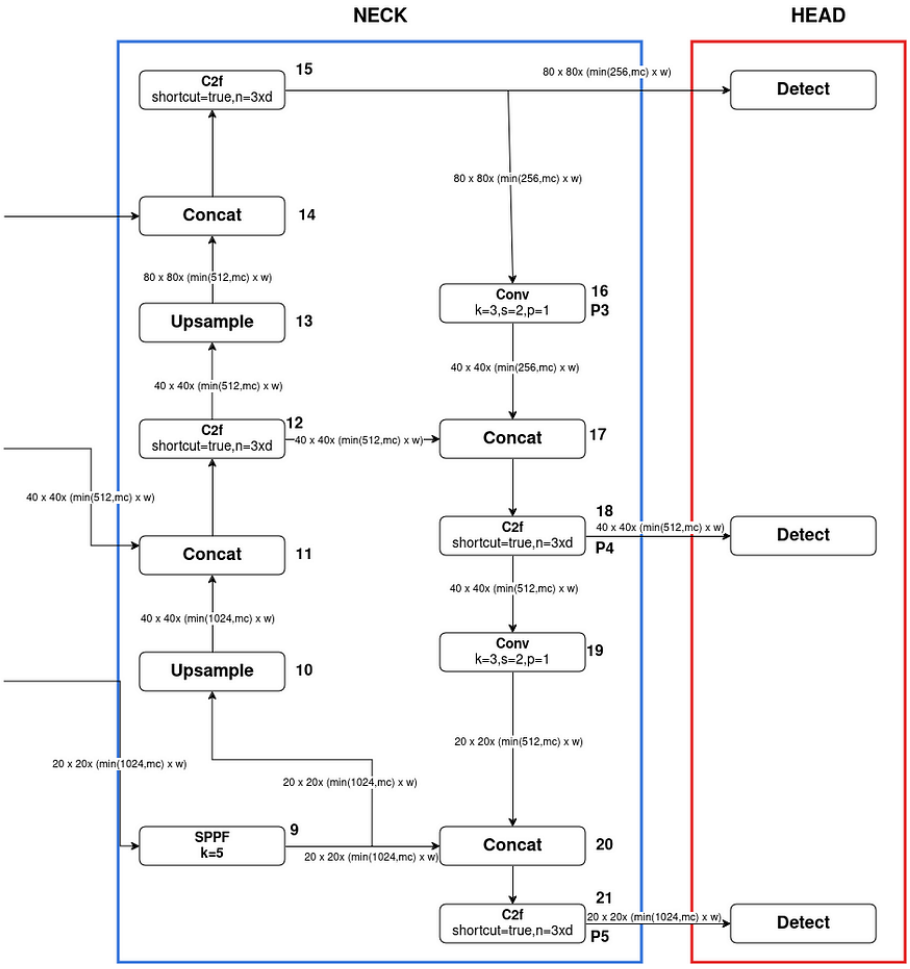
SPPF: Fitur dari bagian 1 diteruskan ke SPPF untuk menggabungkan informasi spasial dari berbagai skala.

Concat: Fitur dari bagian 2 digabungkan (*concatenate*) dengan fitur dari jalur lain untuk memperkaya informasi fitur.

2.9.2. Neck

Neck YOLO v8 menggunakan CLC untuk menghubungkan fitur dari berbagai skala gambar. CLC terdiri dari dua blok, yaitu *Conv* dan *Concat*. Blok *Conv* menggunakan *convolutional layer* untuk mengekstraksi fitur dari fitur yang diumpankan. Blok *Concat* menggabungkan fitur dari berbagai skala gambar.

CLC membantu meningkatkan aliran informasi dalam arsitektur YOLO v8. Hal ini memungkinkan algoritma untuk belajar dari fitur dari berbagai skala gambar dan meningkatkan akurasi deteksi objek.



Gambar 2. 14 Arsitektur Neck

Bagian *neck* dan *head* pada YOLOv8 yang ditunjukkan pada Gambar 2.14 memiliki peran yang sangat penting dalam proses deteksi objek. *Neck* berfungsi untuk menggabungkan fitur-fitur yang telah diekstrak oleh *backbone*, sedangkan *head* bertanggung jawab untuk melakukan prediksi *bounding box* dan kelas objek. Dengan arsitektur yang efisien dan efektif, YOLOv8 mampu mendeteksi objek dengan akurasi yang tinggi dan kecepatan yang *real-time*.

1. Bagian *neck*

   Bertanggung jawab untuk mengambil sampel peta fitur dan menggabungkan fitur yang diperoleh dari berbagai lapisan bagian *Backbone*.

2. Lapisan *upsample* yang ada di bagian *neck*

   Proses *upsampling* pada bagian arsitektur YOLO v8 mengubah ukuran gambar input dari 20 x 20 menjadi 40 x 40. *Upsampling* adalah teknik yang digunakan untuk meningkatkan resolusi gambar atau fitur map, biasanya dengan mengalikan ukuran gambar dengan faktor tertentu. Meningkatkan peta fitur sebanyak dua kali lipat tanpa membuat perubahan apa pun pada saluran keluaran. Proses ini memungkinkan model untuk menggabungkan informasi dari berbagai resolusi, yang penting untuk mendeteksi objek dengan berbagai ukuran dan skala. *Upsample* dalam arsitektur YOLO v8 adalah komponen penting yang memungkinkan peningkatan resolusi fitur map untuk menggabungkan informasi spasial yang lebih baik dan deteksi objek pada berbagai skala. Metode *upsampling* yang digunakan termasuk *nearest neighbor*, *bilinear interpolation*, dan *transpose convolution*, masing-masing dengan cara penerapan yang berbeda. Proses ini meningkatkan kemampuan model untuk mendeteksi objek kecil dan beragam, membuat YOLO v8 menjadi model yang kuat untuk deteksi objek *real-time*.

3. Blok *Concat*

   Dalam arsitektur YOLO v8, *concat* (*concatenation*) adalah proses penggabungan fitur dari berbagai lapisan jaringan untuk menghasilkan representasi fitur yang lebih kaya dan mendetail. Proses ini memungkinkan model untuk memanfaatkan informasi dari berbagai skala dan tingkat detail, yang sangat penting untuk mendeteksi objek dengan ukuran dan bentuk yang berbeda-beda. Bertujuan untuk menjumlahkan saluran keluaran dari blok yang digabungkan tanpa perubahan resolusi apa pun.

4. Bagian *head*

   Bertugas untuk memprediksi kelas dan kotak pembatas objek yang merupakan keluaran akhir yang dihasilkan oleh model deteksi objek.

   a. Blok deteksi pertama di bagian *head* ukuran 80 x 80 (*small objects*)
      Fitur map dengan ukuran 80 x 80 dirancang untuk mendeteksi objek yang lebih kecil. Ukuran *grid* yang lebih besar (banyak sel)

memungkinkan deteksi yang lebih detail dan akurat pada objek kecil yang mungkin tidak terlihat pada resolusi yang lebih rendah. Setiap sel dalam grid 80 x 80 bertanggung jawab untuk mendeteksi objek dalam area kecil dari gambar asli.

b. Blok deteksi kedua di bagian *head*

Fitur map dengan ukuran 40 x 40 digunakan untuk mendeteksi objek berukuran sedang. Ukuran *grid* menengah ini merupakan kompromi antara detail dan cakupan, memungkinkan model untuk menangkap objek yang tidak terlalu kecil tetapi juga tidak terlalu besar. Setiap sel dalam grid 40 x 40 mencakup area yang lebih luas dibandingkan dengan grid 80 x 80, tetapi masih mempertahankan cukup detail untuk mendeteksi objek sedang dengan baik.

c. Blok deteksi ketiga di bagian *head*

Fitur map dengan ukuran 20 x 20 dirancang untuk mendeteksi objek yang lebih besar. Ukuran *grid* yang lebih kecil (lebih sedikit sel) cocok untuk menangani objek besar yang mencakup area yang lebih luas dari gambar. Setiap sel dalam *grid* 20 x 20 mencakup area yang lebih besar, memungkinkan deteksi objek besar dengan efisiensi yang lebih baik.

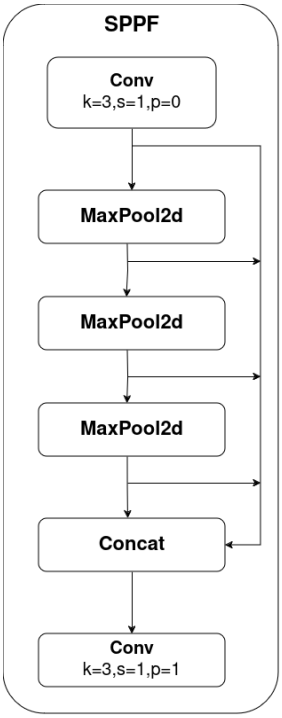
Dengan pendekatan ini, YOLO v8 mampu mendeteksi objek dari berbagai skala dengan akurasi dan efisiensi tinggi, menjadikannya salah satu metode deteksi objek yang sangat efektif

Kesimpulannya, YOLO v8, sebuah evolusi dari keluarga YOLO, mendefinisikan ulang deteksi objek dengan arsitektur bebas jangkar, menyeimbangkan kecepatan dan akurasi di berbagai varian model. Memanfaatkan blok konvolusional dan hambatan, serta fitur inovatif seperti *Spatial Pyramid Pooling Fast*, YOLO v8 secara efisien memproses gambar untuk deteksi waktu nyata. Bagian tulang punggung, leher, dan kepalanya bersinergi untuk mengekstrak fitur, mengambil sampel, dan memprediksi kelas dan kotak pembatas. Dengan keserbagunaannya, YOLO v8 menawarkan beragam model yang memenuhi

beragam kebutuhan, mulai dari inferensi cepat hingga akurasi tinggi. Secara keseluruhan, YOLO v8 mewakili puncak dalam pendeteksian objek, memberdayakan aplikasi dengan kinerja tak tertandingi dan kemudahan penggunaan.

A. *Spatial Pyramid Pooling Fast* (SPPF) *Block*

SPPF *block* merupakan singkatan dari *Spatial Pyramid Pooling - Fast*. Blok ini adalah elemen penting yang dirancang untuk meningkatkan kemampuan model dalam mengumpulkan informasi dari berbagai skala spasial secara efisien dan cepat. SPPF *block* menggabungkan konsep dari *Spatial Pyramid Pooling* (SPP) dengan optimasi untuk kecepatan, menghasilkan blok yang mampu menangkap informasi spasial dari berbagai skala dengan *overhead* komputasi yang minimal.



Gambar 2. 15 Arsitektur *Spatial Pyramid Pooling Fast* (SPPF) *Block*

Pada Gambar 2.15 menunjukkan Blok SPPF terdiri dari blok konvolusional diikuti oleh tiga lapisan *MaxPool2d*. Setiap peta fitur yang dihasilkan dari

lapisan *MaxPool2d* kemudian digabungkan di bagian akhir dan dimasukkan ke blok konvolusional.

Ide dasar dibalik *Spatial Pyramid Pooling* adalah untuk membagi gambar masukan menjadi fitur *grid* dan *pool* dari setiap sel *grid* secara independen, sehingga memungkinkan jaringan untuk menangani gambar dengan ukuran berbeda secara efektif. Intinya, Pengumpulan Piramida Spasial memungkinkan jaringan saraf bekerja dengan gambar dengan resolusi berbeda dengan menangkap informasi multi-skala melalui operasi pengumpulan pada tingkat granularitas berbeda. Hal ini khususnya berguna dalam tugas-tugas seperti pengenalan objek, di mana objek dapat muncul pada skala berbeda dalam suatu gambar.

Meskipun SPP menawarkan keuntungan, namun biaya komputasinya mahal. SPP-*Fast* mengatasi hal ini dengan menggunakan skema pengumpulan yang lebih sederhana. Daripada menggunakan beberapa tingkat pengumpulan dengan ukuran kernel berbeda, SPP-*Fast* mungkin menggunakan satu kernel berukuran tetap untuk pengumpulan, sehingga mengurangi jumlah komputasi yang diperlukan. SPP-*Fast* menawarkan *trade-off* antara akurasi dan kecepatan.

a. MaxPool2d Layer

Lapisan pengumpulan digunakan untuk menurunkan sampel dimensi spasial volume masukan, mengurangi kompleksitas komputasi jaringan dan mengekstraksi fitur dominan. Pengumpulan maksimum adalah jenis operasi pengumpulan tertentu yang mana, untuk setiap wilayah di tensor masukan, hanya nilai maksimum yang dipertahankan, dan nilai lainnya dibuang.

Dalam kasus *MaxPool2d*, penggabungan diterapkan pada dimensi tinggi dan lebar tensor masukan. Lapisan ini ditentukan dengan menentukan parameter seperti ukuran kernel *pooling* dan langkahnya. Ukuran kernel menentukan luas spasial setiap wilayah pengumpulan, dan langkahnya menentukan ukuran langkah antara wilayah pengumpulan yang berurutan.

Setelah proses *Spatial Pyramid Pooling Fast* (SPPF) dalam arsitektur YOLO v8, jalur fitur map terbagi menjadi dua, yaitu jalur menuju operasi *concat* (penggabungan) dan jalur lainnya menuju operasi *upsample*.

1. Jalur *concatenation (Concat)*

   Salah satu jalur dari output SPPF langsung menuju ke operasi *concat*. Pada operasi ini, fitur map dari SPPF digabungkan dengan fitur map dari *layer* sebelumnya yang memiliki resolusi yang sama. Penggabungan ini bertujuan untuk mengintegrasikan informasi dari berbagai lapisan, memperkaya representasi fitur yang ada. Contohnya Fitur map berukuran 20 x 20 dari SPPF digabungkan dengan fitur map berukuran 20 x 20 dari *layer* sebelumnya.
2. Jalur *upsample*

   Jalur lainnya dari *output* SPPF menjalani operasi *upsample*. *Upsampling* adalah proses memperbesar dimensi fitur map dengan meningkatkan resolusi (misalnya, dari 20 x 20 menjadi 40 x 40). Operasi ini dilakukan menggunakan metode interpolasi untuk menambah informasi detail pada skala yang lebih tinggi. Setelah *upsample*, fitur map yang diperbesar digabungkan dengan fitur map dari *layer* sebelumnya yang memiliki resolusi lebih tinggi (misalnya, 40 x 40). Penggabungan ini memungkinkan integrasi informasi dari berbagai skala, membantu model untuk mengenali objek pada resolusi yang lebih tinggi.

Dengan membagi jalur menjadi *concat* dan *upsample*, YOLO v8 dapat menangkap informasi dari berbagai skala dan resolusi, meningkatkan akurasi deteksi untuk objek kecil, sedang, dan besar. Penggabungan fitur map dari berbagai lapisan memastikan bahwa informasi penting tidak hilang dan diperkaya melalui proses deteksi. *Upsampling* memungkinkan model untuk menambahkan detail pada skala yang lebih tinggi, memastikan objek kecil tidak terlewatkan.

Dengan demikian, pembagian jalur setelah SPPF menjadi *concat* dan *upsample* dalam arsitektur YOLO v8 memainkan peran penting dalam

meningkatkan kemampuan model untuk mendeteksi objek secara akurat pada berbagai skala dan resolusi.

B. Upsample

Upsample adalah blok yang digunakan untuk meningkatkan resolusi gambar *input*. Blok ini menggunakan *bilinear interpolation* untuk meningkatkan resolusi gambar dengan faktor tertentu. *Upsampling* digunakan dalam *neck* YOLO v8 untuk menggabungkan fitur dari berbagai skala spasial.

Fitur dari SPPF diumpankan ke blok *upsample* untuk meningkatkan resolusi gambar. *Upsampling* dilakukan dengan faktor tertentu untuk menggabungkan fitur dari berbagai skala spasial.

C. Concatenate

Concat adalah operasi yang digunakan untuk menggabungkan dua atau lebih tensor. Operasi ini digunakan dalam *neck* YOLO v8 untuk menggabungkan fitur dari berbagai skala spasial. Operasi *concat* memungkinkan algoritma untuk belajar dari fitur dari berbagai skala dan meningkatkan akurasi deteksi objek.

Fitur dari SPPF dan *upsample* digabungkan menggunakan operasi *concat*. Operasi *concat* memungkinkan algoritma untuk belajar dari fitur dari berbagai skala.

D. C2F (*Channel to Feature*)

C2F adalah blok yang digunakan untuk mengubah dimensi *channel* dari tensor. Blok ini digunakan dalam *neck* YOLO v8 untuk memastikan bahwa dimensi *channel* dari tensor yang digabungkan sama.

Fitur yang digabungkan diumpankan ke blok C2F untuk memastikan dimensi *channel* yang sama. C2F mengubah dimensi *channel* dari tensor agar sesuai dengan dimensi *channel* dari tensor lain yang akan digabungkan.

E. Convolution

Convolution adalah blok yang digunakan untuk mengekstraksi fitur dari gambar *input*. Blok ini terdiri dari *convolutional layer* yang menggunakan *kernel size* dan *filter size* tertentu. *Convolution* digunakan dalam *neck* YOLO v8 untuk mengekstraksi fitur dari gambar *input* pada berbagai skala spasial.

Fitur dari C2F diumpankan ke blok *convolution* untuk mengekstraksi fitur akhir. *Convolution block* menggunakan *kernel size* dan *filter size* tertentu untuk mengekstraksi fitur yang relevan untuk deteksi objek.

Neck YOLO v8 memiliki dua percabangan:

1. Percabangan pertama menggabungkan fitur dari *stage* 1, 2, dan 3 FPN. Percabangan ini digunakan untuk mendeteksi objek pada skala kecil, menengah, dan besar.
2. Percabangan kedua menggabungkan fitur dari *stage* 2 dan 3 FPN. Percabangan ini digunakan untuk mendeteksi objek pada skala menengah dan besar.

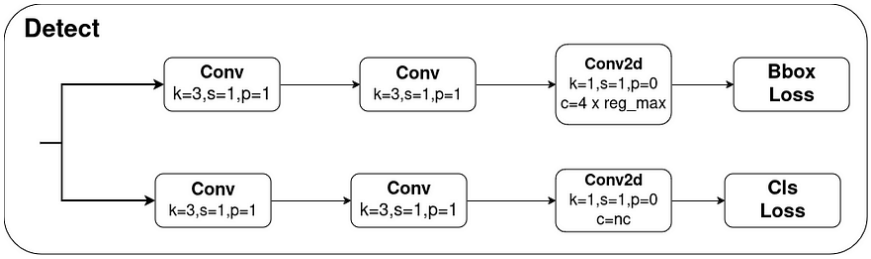
Percabangan ini memungkinkan algoritma untuk fokus pada deteksi objek pada skala yang berbeda. Percabangan pertama fokus pada deteksi objek kecil, sedangkan percabangan kedua fokus pada deteksi objek besar.

2.9.3. Head

YOLO v8 menggunakan tiga *head* yang berbeda untuk mendeteksi objek pada skala yang berbeda. *Head* ini menggunakan algoritma deteksi objek yang disebut CSP. CSP terdiri dari dua blok, yaitu *Bottleneck* dan CSP *Route*. Blok *bottleneck* menggunakan *convolutional layer* untuk mengekstraksi fitur dari fitur yang diumpankan. Blok CSP *Route* menggabungkan fitur dari dua jalur yang berbeda. CSP membantu meningkatkan akurasi dan presisi deteksi objek. Hal ini memungkinkan algoritma untuk belajar dari fitur dari dua jalur yang berbeda dan meningkatkan kemampuannya untuk membedakan antara objek yang berbeda.

A. Detect Block

Detect Block adalah komponen kunci yang bertanggung jawab untuk melakukan tugas utama dari jaringan ini, yaitu mendeteksi objek dalam citra *input*. *Detect block* menggabungkan informasi dari berbagai lapisan jaringan sebelumnya untuk menghasilkan prediksi *bounding box*, kelas objek, dan skor keyakinan untuk setiap objek yang terdeteksi. *Detect block* berfungsi untuk mengubah fitur peta (*feature map*) yang dihasilkan oleh *backbone* dan *head* jaringan menjadi prediksi yang dapat digunakan untuk deteksi objek.



Gambar 2. 16 Arsitektur *Detect Block*

Bagian *detect* yang ditunjukkan oleh Gambar 2.16 merupakan bagian akhir dari jaringan saraf tiruan YOLOv8 yang bertanggung jawab langsung dalam proses deteksi objek. Bagian ini menerima *input* dari bagian *neck*, yang merupakan gabungan dari fitur-fitur yang diekstrak dari berbagai tingkatan resolusi gambar.

Setelah fitur diekstraksi oleh *backbone* dan diproses oleh *neck*, *head* jaringan bertugas untuk menghasilkan prediksi akhir. Di sini, klasifikasi objek dilakukan dengan memberikan probabilitas kelas untuk setiap *bounding box* yang terdeteksi. Proses ini mencakup:

1. *Bounding Box Regression*

   Menentukan lokasi dan ukuran *bounding box* untuk setiap objek yang terdeteksi.
2. *Objectness Score*

   Memberikan *confidence score* yang menunjukkan kemungkinan adanya objek di dalam *bounding box*.

3. *Class Probability*

   Memberikan probabilitas untuk setiap kelas objek dalam *bounding box*, berdasarkan fitur yang diekstraksi dan digabungkan.

Detect Block bertanggung jawab untuk mendeteksi objek. Tidak seperti YOLO versi sebelumnya, YOLO v8 adalah model bebas jangkar yang berarti model ini memprediksi secara langsung pusat suatu objek, bukan *offset* dari kotak jangkar yang diketahui. Deteksi bebas jangkar mengurangi jumlah prediksi kotak, yang mempercepat langkah pasca-pemrosesan rumit yang menyaring kandidat deteksi setelah inferensi.

Detect Block berisi dua trek. Jalur pertama untuk prediksi kotak pembatas dan jalur kedua untuk prediksi kelas . Kedua trek berisi dua blok konvolusional diikuti oleh satu lapisan Konv2d yang masing-masing memberikan kerugian kotak batas dan kerugian kelas.

1. Cabang Prediksi *Bounding Box*:

   Conv (k=3, s=1, p=1):

   Lapisan konvolusi pertama dengan kernel 3x3, *stride* 1, dan *padding* 1 digunakan untuk mengekstrak fitur-fitur yang lebih relevan untuk memprediksi *bounding box*.

   Conv (k=3, s=1, p=1):

   Lapisan konvolusi kedua dengan parameter yang sama berfungsi untuk menyaring fitur lebih lanjut.

   Conv2d (k=1, s=1, p=0, c=4*reg_max):

   Lapisan konvolusi 1x1 ini menghasilkan tensor dengan jumlah *channel* yang sama dengan 4 kali jumlah *anchor box* maksimum (*reg_max*). Setiap 4 *channel* akan memprediksi empat nilai: tx, ty, tw, dan th yang merepresentasikan *offset* dari pusat *bounding box* yang diprediksi terhadap pusat *anchor box*.

   Bounding Box Loss:

   Hasil prediksi *bounding box* akan dibandingkan dengan *ground truth* (*bounding box* yang sebenarnya) menggunakan fungsi *loss* seperti *CIoU*

loss. Tujuannya adalah untuk meminimalkan perbedaan antara prediksi dan *ground truth*.

2. Cabang Prediksi Kelas:

   Conv (k=3, s=1, p=1):

   Dua lapisan konvolusi pertama memiliki fungsi yang sama seperti pada cabang *bounding box*, yaitu mengekstrak fitur yang relevan untuk memprediksi kelas objek.

   Conv2d (k=1, s=1, p=0, c=nc):

   Lapisan konvolusi 1x1 ini menghasilkan tensor dengan jumlah *channel* yang sama dengan jumlah kelas (nc). Setiap channel akan memprediksi probabilitas objek termasuk dalam kelas tertentu.

   Class Loss:

   Hasil prediksi kelas akan dibandingkan dengan *ground truth* (label kelas) menggunakan fungsi *loss* seperti *cross-entropy loss*. Tujuannya adalah untuk memaksimalkan probabilitas kelas yang benar.

Arsitektur YOLO v8 menggunakan berbagai teknik pemrosesan dan penggabungan fitur untuk mencapai deteksi objek yang akurat dan efisien.

1. Pembagian dan Penggabungan Fitur

   Feature Splitting: Dalam banyak tahap, fitur yang diekstraksi dari gambar dibagi menjadi beberapa jalur pemrosesan untuk mengolah informasi secara paralel. Ini membantu model menangkap berbagai aspek dari data input, seperti detail kecil dan konteks luas.

   Concatenation: Fitur dari berbagai jalur digabungkan kembali untuk memperkaya representasi fitur. Penggabungan ini memastikan bahwa model memiliki informasi yang cukup dari berbagai skala dan perspektif.

2. *Multi-Scale Processing*

   Different Scales: Arsitektur YOLO v8 memproses gambar pada berbagai skala (misalnya, 80 x 80, 40 x 40, 20 x 20). Setiap skala menangani objek dengan ukuran yang berbeda, memastikan bahwa model dapat mendeteksi objek kecil maupun besar dengan efektif.

Pyramid Structures: Penggunaan struktur seperti *Spatial Pyramid Pooling* (SPP) membantu menggabungkan informasi dari berbagai skala, memperkuat model dalam memahami konteks yang lebih luas dari gambar.

3. *Repeated Blocks*

   Convolutional Blocks: Banyak proses *convolutional* (conv) diulang dalam arsitektur untuk memperdalam pemrosesan fitur dan menangkap pola yang lebih kompleks.

   C2f Blocks: Blok C2f (*Convolutional Cross Stage Partial Networks*) diulang untuk memperkaya representasi fitur dengan memproses informasi secara bertahap.

4. *Feature Fusion*

   Upsample and Downsample: Proses *upsampling* dan *downsampling* memungkinkan model mengubah resolusi gambar untuk menggabungkan fitur dari berbagai skala. Ini membantu dalam menangani objek dari berbagai ukuran dalam satu jaringan.

   Skip Connections: Koneksi lompat (*shortcut*) menghubungkan lapisan yang jauh terpisah dalam jaringan, memungkinkan aliran informasi yang lebih efisien dan mengurangi masalah *vanishing gradient*.

5. *Head for Detection*

   Multi-Output: Arsitektur YOLOv8 menghasilkan beberapa keluaran (*output*) pada berbagai skala resolusi untuk deteksi objek. Ini memaksimalkan kemampuan model untuk mendeteksi objek dengan ukuran yang berbeda secara simultan.

Kompleksitas arsitektur YOLO v8 adalah hasil dari desain yang cermat untuk menangani berbagai tantangan dalam deteksi objek, seperti skala objek yang berbeda, konteks gambar yang luas, dan efisiensi pemrosesan. Dengan memahami tujuan dan struktur dari proses pembagian dan penggabungan fitur, hasilnya model YOLO v8 memiliki keunggulan dalam mendeteksi objek secara real-time dengan akurasi tinggi.

2.9.3.1 Confussion Matrix

Confusion Matrix adalah suatu pengukuran kinerja atau performa dalam menyelesaikan masalah klasifikasi *machine learning*, dimana hasil *output* dapat berupa dua kelas atau lebih. *Confusion matrix* merupakan suatu alat analisis prediktif yang menampilkan dan membandingkan nilai sebenarnya dengan nilai hasil prediksi model. Prediksi model yang dapat digunakan untuk mendapatkan hasil matriks evaluasi yakni Akurasi, *Precision*, *Recall* dan F1 *Score* [28]. Berikut dapat dilihat pada Gambar 2.17 merupakan 4 kombinasi berbeda dari nilai prediksi dan nilai sebenarnya.

		Nilai Sebenarnya	
		1 (*Positive*)	0 (*Negative*)
Nil ai Prediksi	1 (*Positive*)	TP (*True Positive*)	FP (*False Negative*)
	0 (*Negative*)	FN (*False Negative*)	TN (*True Negative*)

Gambar 2. 17 Confusion Matrix

Dengan melihat nilai TP, FP, FN, dan TN yang ditunjukkan pada Gambar 2.17, dapat disimpulkan bahwa nilai *confusion matrix* dapat mengetahui seberapa baik model dalam melakukan klasifikasi.

TP (*True Positive*) : Jumlah data yang bernilai positif dan diprediksi benar sebagai positif.

FP (*False Negative*) : Jumlah data yang bernilai negatif tetapi diprediksi sebagai positif.

FN (*False Negative*) : Jumlah data yang bernilai positif tetapi diprediksi sebagai negatif.

TN (*True Negative*) : Jumlah data yang bernilai negatif dengan prediksi benar sebagai negatif.

Berikut adalah rumus perhitungan akurasi:

$$\text{Akurasi} = \frac{TP+TN}{TP+TN+FP+FN} \; X \; 100\% \quad \text{.....(6)}$$

2.9.3.2 Average Precision (AP)

Average Precision (AP) adalah standar *de facto* untuk evaluasi kinerja dalam deteksi objek. Deteksi objek gambar statis yang populer, deteksi objek video, dan penelitian deteksi objek video *online* terutama melaporkan hasil AP dan *mean*-AP, AP tampaknya menjadi satu-satunya kriteria yang digunakan untuk membandingkan metode deteksi objek [29]. Dalam perhitungan AP dapat menggunakan *N-Point Inrterpolation,* untuk nilai *recall* dilambangkan dengan $R_r(n)$ dan sedangkan untuk nilai *precision* dilambangkan dengan $Pr_{interp}(R_r n)$. Untuk persamaan dari $R_r(n)$ dapat dilihat pada persamaan 7.

$$R_r(n) = {}^{N}\!\_\_\_{}^{N^{-}-n_1} \;, \qquad n = 1, 2, \dots, N. \quad \text{..... (7)}$$

Sehingga untuk persamaan AP menjadi :

$$AP \quad = \frac{1}{} \sum N \qquad {}^{N}{}_{n=1} \, Pr_{interp}(R_r(n)). \quad \text{..... (8)}$$

Persamaan tersebut merupakan persamaan untuk mencari nilai AP menggunakan *N-Point Interpolation*. Biasanya nilai *N* yang dipakai pada mencari hasil AP yakni *N* = 11.

2.9.3.3 Akurasi

Akurasi mengukur seberapa sering model membuat prediksi yang benar secara keseluruhan. Ini dihitung sebagai rasio jumlah prediksi yang benar (baik positif maupun negatif) dengan total jumlah prediksi. Akurasi dapat menyesatkan jika

dataset tidak seimbang, misalnya jika jumlah kelas positif jauh lebih sedikit dari kelas negatif. Persamaan akurasi dapat dilihat dari persamaan (9).

Akurasi= $\frac{TP}{TP+FN}$ x 100%.....(9)

2.9.3.4 Precision

Precision adalah suatu rasio dari jumlah objek dengan benar atau *true positif* dibandingkan dengan seluruh data yang diprediksi positif. Nilai *precision* akan semakin rendah jika nilai dari *False Positive* (FP) semakin besar dan begitupun sebaliknya [30]. Persamaan presisi dapat dilihat pada Persamaan (9).

Precision = $\frac{TP}{TP+FP}$ x 100% …….(9)

2.9.3.5 Recall

Recall merupakan rasio dari jumlah objek terdeteksi dengan benar atau *True Positive* dibandingkan dengan seluruh data yang positif, *recall* yang memiliki nilai yang tinggi menunjukkan bahwa sistem dapat mengklasifikasi kelas objek dengan benar [30].

Persamaan *recall* dapat dilihat pada Persamaan (10).

Recall = $\frac{TP}{TP+FN}$ x 100% ……..(10)

2.9.3.6 F1 Score

F1 *Score* adalah suatu perbandingan rata-rata dari nilai presisi dan *recall*. F1 *Score* memiliki nilai tertinggi sebesar 1 dan terendah sebesar 0. Nilai F1 *Score* ini semakin mendekati 1 menunjukkan bahwa kinerja sistem telah baik [30]. Persamaan F1 *Score* dapat dilihat pada Persamaan (11):

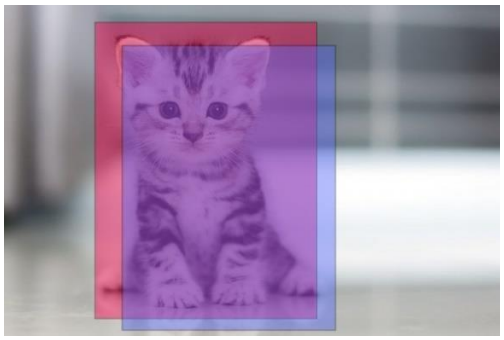
F1 Score = 2 x $\frac{Recall\ x\ Presisi}{Recall+Presisi}$ ……(11)

2.9.3.7 Intersection Over Union (IoU)

Intersection over union adalah rasio antara area perpotongan (*intersection*) dari prediksi *bounding box* dengan *ground truth bounding box* terhadap area gabungan (*union*) dari kedua *bounding box* tersebut. *Intersection over union* digunakan untuk

menghitung suatu luas area yang berpotongan kemudian membaginya dengan luas area gabungan antara 2 *bounding box*. Pada umumnya nilai IoU ini dipakai untuk menentukan seberapa baik performa dari *bounding box* yang diprediksi dengan luas objek sesungguhnya pada suatu citra [31]. Perhitungan IoU dapat di lihat pada Persamaan (12):

$$IoU = \frac{Area\ irisan}{Area\ Gabungan} \quad \text{.....(12)}$$



Gambar 2. 18 Contoh tampilan deteksi IOU

Gambar 2.18 merupakan tampilan ilustrasi dari deteksi IoU pada objek kucing, dapat dilihat terdapat dua *bounding box*. *Bounding box* merah adalah *ground truth* dan *bounding box* berwarna biru adalah *bounding box* prediksi. Sehingga jika ingin mendapat nilai IoU dari objek kucing tersebut dapat diimplementasikan Persamaan 12.

Intersection over union (IoU) adalah metrik penting yang digunakan selama *training* model YOLO v8 untuk mengevaluasi kualitas prediksi *bounding box*. Dengan menghitung IoU antara prediksi *bounding box* dan *ground truth bounding box*, model dapat secara efektif memperbaiki kesalahannya. IoU membantu dalam memastikan bahwa prediksi *bounding box* akurat dan sesuai dengan objek yang sebenarnya, sehingga meningkatkan keseluruhan performa model dalam tugas deteksi objek sampah organik dan anorganik.

2.9.3.8 Loss Function

Loss function (fungsi kerugian) pada pelatihan model YOLO (*You Only Look Once*) adalah fungsi yang digunakan untuk mengukur seberapa baik prediksi model sesuai dengan target atau *ground truth* [31]. Dalam konteks YOLO, *loss function* dirancang untuk menilai kesalahan dalam prediksi lokasi dan kelas objek. Penggunaan *Mean Squared Error* (MSE) sebagai salah satu metode perhitungan *loss function* sangat umum dalam banyak model *deep learning*, termasuk dalam deteksi objek seperti YOLO. MSE adalah salah satu *loss function* paling sederhana dan mudah diimplementasikan. Ini menghitung rata-rata kuadrat dari selisih antara nilai prediksi dan nilai aktual (*ground truth*). Kesederhanaan ini menjadikannya pilihan utama untuk tugas-tugas seperti regresi, termasuk dalam memprediksi koordinat *bounding box* pada tugas deteksi objek. Karena MSE mengkuadratkan *error* (selisih antara prediksi dan *ground truth*), ini secara otomatis memberikan penalti yang lebih besar untuk kesalahan besar. Dengan kata lain, MSE lebih sensitif terhadap *outliers* atau prediksi yang jauh dari *ground truth*, yang berguna dalam mendeteksi *bounding box* yang jauh dari target [32]. Perhitungan *Mean Square Error* dapat dilihat dari persamaan 13 berikut

$$Mean\ Square\ Error = \frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2 \quad \text{.....(13)}$$

Ada beberapa komponen utama dalam loss function untuk YOLO, termasuk:

1. ***Localization Loss***

   Bagian dari *loss function* yang mengukur seberapa akurat prediksi lokasi (kotak pembatas) dibandingkan dengan *ground truth*. Ini sering dihitung menggunakan metrik seperti *mean squared error* (MSE) atau *smooth* L1 *loss* antara koordinat prediksi dan *ground truth*.

$$loss_{ij}^{xywh} = \lambda_{coord}\sum_{i=0}^{S^2}\sum_{j=0}^{B}L_{ij}^{obj}\left[(X_{ij} - \hat{X}_{ij})^2 + (y_{ij} - \hat{y}_{ij})^2\right] +$$

$$\sum_{i=0}^{S^2}\sum_{j=0}^{B}\left[(\sqrt{W_{ij}} - \sqrt{\hat{W}_{ij}})^2 + (\sqrt{h_{ij}} - \sqrt{\hat{h}_{ij}})^2\right] \text{.....(14)}$$

$$L_{ij}^{obj} = \begin{cases} 1\ jika\ C_{ij} = 1 \\ 0 \end{cases}$$

$L_{ij}^{noobj} = \begin{cases} 1\ jika\ max_{ij} dari\ IOU < tresh\ dan\ C_{ij} = 0 \\ 0 \end{cases}$

$\sum_{j=0}^{B} =$ Jumlah *loss* dari setiap *anchor box*

$\sum_{i=0}^{S^2} =$ Jumlah *loss* dari setiap S x S *grid*

$L_{ij}^{obj} =$ Indikasi objek dengan nilai 0 / 1

λ_{coord} = Konstanta untuk mengatur *loss* pada prediksi *bounding box* dengan nilai 5

λ_{noobj} = Konstanta untuk mengatur *loss* pada tingkat kepercayaan untuk *grid* yang tidak memiliki objek dengan nilai 0.5

$X_{ij}, Y_{ij}, W_{ij}, h_{ij}$ = Nilai *bounding box* dari hasil prediksi

$\hat{X}_{ij}, \hat{Y}_{ij} \hat{W}_{ij} \hat{h}_{ij}$ = Nilai *bounding box* dari data masukan

2. ***Confidence Loss***

Bagian dari *loss function* yang menilai seberapa yakin model dalam memprediksi keberadaan objek di dalam kotak pembatas. Ini mencakup penalti untuk kotak pembatas yang mengandung objek (*true positive*) dan yang tidak (*false positive*).

$loss_{ij}^{c} = \sum_{i=0}^{S^2} \sum_{j=0}^{B} L_{ij}^{obj} (IOU - \hat{C}_{ij})^2 + \lambda_{noobj} \sum_{i=0}^{S^2} \sum_{j=0}^{B} L_{ij}^{obj} (0 - \hat{C}_{ij})^2$ (15)

$\hat{C}_{ij}$ = Nilai kepercayaan dari objek dari data masukan

3. ***Classification Loss***

Jika YOLO digunakan untuk tugas klasifikasi objek, komponen ini menilai kesalahan dalam prediksi kelas objek. Ini sering menggunakan *cross-entropy loss* antara distribusi probabilitas kelas prediksi dan *ground truth*.

$loss_{ij}^{p} = \sum_{i=0}^{S^2} L_{ij}^{obj} \sum_{classes} (p_{ij}(c) - \hat{p}_{ij}(c))^2$ ……(16)

$p_{ij}(c)$ = Probabilitas terdapat *class* hasil prediksi

$\hat{p}_{ij}(c)$ = Probabilitas terdapat *class* sebenarnya

$$Smooth\ L1\ Loss\ (x) = \begin{cases} 0.5\,.x^2 & if\ |x| < 1 \\ |x| - 0.5 & otherwise \end{cases} \text{.....(17)}$$

Total Loss (L_{total}) adalah kombinasi dari ketiga komponen *loss*:

$$loss_{ij} = loss_{ij}^{xywh} + loss_{ij}^{c} + loss_{ij}^{p} \text{ .....(18)}$$

Loss function dalam YOLO dirancang untuk menyeimbangkan antara prediksi lokasi, keberadaan objek, dan klasifikasi objek. Tujuan dari pelatihan model YOLO adalah untuk mengoptimalkan *loss function* ini sehingga model dapat membuat prediksi yang akurat dan tepat dalam mendeteksi dan mengklasifikasikan objek dalam gambar.

Selama proses *training*, *loss function* berperan sebagai panduan yang membantu model belajar dari kesalahan dan memperbaiki prediksinya. Dengan menghitung kesalahan dalam prediksi *bounding box, confidence score*, dan klasifikasi objek, *loss function* memastikan bahwa model semakin baik dalam mendeteksi dan mengklasifikasikan objek dalam gambar. Kombinasi dari berbagai komponen *loss function* dan penggunaan teknik optimasi yang tepat memungkinkan YOLO v8 untuk mencapai performa yang optimal dalam tugas deteksi objek sampah organik dan anorganik.

2.9.3.9 Mean Average Precisision (mAP)

Mean average precission atau mAP adalah suatu matriks yang mengevaluasi kinerja dari model deteksi objek, dan mengukur sebarapa bagus performasi dari *weights file* hasil *training data* [30]. Nilai mAP didapatkan dari setiap nilai presisi *item* relevan yang dihasilkan dan menggunakan nilai 0 untuk *item* relevan yang tidak dihasilkan oleh sistem [31].

Persamaan mAP dapat di lihat pada Persamaan (18).

$$mAP = \frac{1}{-}\sum c\ ^{c}{}_{t=1}\ AP_i \text{ .....(19)}$$

2.9.3.10 Non-Maximum Suppression

Non-max supression digunakan untuk menyeleksi *bounding box* yang muncul *overload* atau berlebihan pada objek yang sama dengan membandingkan nilai *confidence* masing-masing *bounding box*, hanya nilai *confidence* yang paling tinggi dan yang akan dipertahankan**.** *Non Max Suppression* membuat seleksi berdasarkan IoU dengan mengurangi *bounding box* yang mucul secara berlebihan [31].

2.9.3.11 Weights

Weights pada model YOLO v8 dihasilkan melalui proses *training* menggunakan dataset yang disediakan. Berikut merupakan proses penghasilan *weights* pada YOLO v8

1. *Initialization* (Inisialisasi):

   Awal *training*: *Weights* pada model YOLO v8 diinisialisasi secara acak atau berdasarkan model *pretrained* (misalnya, dari model YOLO v8 yang telah dilatih sebelumnya pada dataset besar seperti COCO).

   Pretrained weights: Jika menggunakan *pretrained weights*, model sudah memiliki kemampuan dasar dalam mendeteksi objek, yang akan disesuaikan dengan dataset spesifik yang digunakan.

2. *Forward pass* (Propagasi Maju):

   Input: Dataset gambar yang telah diproses (augmentasi, normalisasi, dll.) dimasukkan ke dalam model.

   Proses Konvolusi: Gambar-gambar ini melewati berbagai lapisan konvolusi, *pooling*, dan aktivasi dalam arsitektur YOLO v8 (*backbone*, *neck*, dan *head*).

   Prediksi: Model menghasilkan prediksi *bounding box* dan klasifikasi objek berdasarkan *current weights*.

3. *Loss calculation* (Perhitungan *loss*):

   Loss function: Perbedaan antara prediksi model dan *ground truth* (label sebenarnya) dihitung menggunakan fungsi *loss*, yang biasanya mencakup:

   Bounding box loss: Mengukur seberapa baik *bounding box* yang diprediksi cocok dengan *bounding box ground truth* (misalnya, menggunakan IoU).

   Classification loss: Mengukur akurasi klasifikasi objek dalam *bounding box*.

   Objectness loss: Mengukur kepercayaan model bahwa sebuah *bounding box* berisi objek yang relevan.

4. *Backward pass* (Propagasi Mundur):

   Gradient calculation: Gradien dari fungsi *loss* terhadap *weights* model dihitung menggunakan algoritma *backpropagation*.

   Gradient descent: *Weights* model diperbarui menggunakan *optimizers* seperti SGD (*Stochastic Gradient Descent*) atau Adam, yang mengurangi nilai *loss* dengan menyesuaikan *weights* berdasarkan gradien yang dihitung.

   Learning rate: Faktor *learning rate* mengontrol seberapa besar perubahan *weights* pada setiap iterasi.

5. *Iteration* (Iterasi):

   Epoch: Proses ini diulang untuk setiap *epoch*, di mana satu *epoch* adalah satu putaran penuh melalui seluruh dataset.

   Batch training: Dataset biasanya dibagi menjadi beberapa batch untuk efisiensi memori dan komputasi, sehingga proses *forward* dan *backward pass* dilakukan per *batch*.

6. *Convergence* (Konvergensi):

   Convergence check: Proses *training* berlangsung hingga model mencapai konvergensi, yaitu ketika perbaikan pada fungsi *loss* menjadi minimal atau stabil.

   Early stopping: Beberapa metode seperti *early stopping* dapat digunakan untuk menghentikan *training* jika tidak ada perbaikan signifikan dalam jangka waktu tertentu.

Setelah proses *training* selesai, *weights* yang telah dioptimalkan disimpan sebagai file (misalnya, yolov8l.pt). File ini berisi parameter dari semua lapisan dalam model YOLO v8 yang telah disesuaikan berdasarkan dataset yang digunakan. *Weights* ini digunakan untuk melakukan inferensi pada data baru, memungkinkan model untuk mendeteksi dan mengklasifikasikan objek berdasarkan pengetahuan yang diperoleh selama *training*.

Terdapat cara perhitungan untuk mendapatkan *weights* selama proses *training* model *neural network*, termasuk YOLO v8. Proses ini melibatkan beberapa langkah matematika yang melibatkan *forward pass*, *backward pass*, dan *update weights*. Berikut adalah langkah-langkah tersebut:

1. *Forward pass* (Propagasi maju)

   Pada *forward pass*, data *input* (misalnya, gambar) melewati lapisan-lapisan jaringan *neural*, dan setiap lapisan melakukan operasi matematis untuk menghasilkan *output* (prediksi).

   Didapatkan jaringan *neural* sederhana dengan satu lapisan konvolusi diikuti oleh aktivasi SiLU (*Swish Linear Unit*):

   $$Z = Conv2D(X, W, b)$$

   $$A = SiLU(Z)$$

   Dimana X adalah *input*, W adalah *weights*, b adalah *bias*, Z adalah *output* dari konvolusi, dan A adalah *output* dari aktivasi

2. *Loss calculation* (Perhitungan *loss*)

   Setelah mendapatkan *output* dari *forward pass*, Kemudian hitung seberapa jauh prediksi model dari *ground truth* menggunakan fungsi *loss*.

   Didapatkan *Mean Squared Error* (MSE) sebagai fungsi *loss*:

   $$\text{Loss} = \frac{1}{n}\sum_{i=1}^{4}(\hat{y}_i - y_i)^2 \quad \text{.....(20)}$$

   y_i adalah *ground truth*

   $\hat{y}_i$ adalah prediksi model

3. *Backward pass* (Propagasi mundur)

   Pada *backward pass*, dihitung gradien dari fungsi *loss* terhadap setiap parameter (*weights* dan *bias*) dalam jaringan menggunakan metode *backpropagation*.

   Jika menghitung gradien dari *loss* terhadap *weights*

   $$\frac{\partial Loss}{\partial Z} = \frac{\partial Loss}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial Z} \cdot \frac{\partial Z}{\partial W} \quad \text{.....(21)}$$

   $\frac{\partial Loss}{\partial Z}$ adalah gradien *loss* terhadap *output*

   $\frac{\partial \hat{y}}{\partial Z}$ adalah gradien *output* terhadap hasil konvolusi

   $\frac{\partial Z}{\partial W}$ adalah gradien hasil konvolusi terhadap *weights*

4. *Weight update* (Pembaruan *weights*)

   Setelah mendapatkan gradien, *weights* diperbarui menggunakan algoritma optimasi seperti *Stochastic Gradient Descent* (SGD) atau Adam.

   Misalkan digunakan SGD:

   $$W_{baru} = W_{lama} - \eta \cdot \frac{\partial Loss}{\partial W} \quad \text{.....(22)}$$

   W_{baru} adalah *weights* yang diperbarui

   W_{lama} adalah *weights* saat ini

η adalah *learning rate*

$\frac{\partial Loss}{\partial W}$ adalah gradien

2.10 Pra Proses Data

Praproses merupakan suatu fase utama yang dapat menentukan kualitas untuk proses berikutnya. Tujuan dari fase pra proses ini ialah untuk memperoleh data siap olah untuk diproses oleh *data mining* dari data awal yang masih berupa data tekstual [32]. Praproses dapat dilakukan seperti mengubah ukuran (*resize*) dataset yang berupa sebuah citra. Tujuan dari *resize* yakni untuk mempermudah dan mempercepat citra ketika diproses.

2.11 Python

Bahasa pemrograman Python adalah bahasa pemrograman yang dibuat oleh Guido van Rossum dari Amsterdam, Belanda. Pada awalnya, pembuatan bahasa pemrograman ini adalah untuk bahasa skrip tingkat tinggi pada sistem operasi terdistribusi amoeba. Bahasa pemrograman ini menjadi umum digunakan untuk kalangan *engineer* seluruh dunia dalam pembuatan perangkat lunaknya, bahkan perusahaan besar seperti Google, NASA, Instagram, Youtube, dan Spotify menggunakan Python sebagai pembuat perangkat lunak komersial. Python banyak digunakan untuk membuat berbagai macam program, seperti program CLI, GUI (*desktop*), aplikasi Mobile, Web, IoT, game, program *hacking*, dan sebagainya [33].

2.12 Penelitian Terkait

Penelitian ini tidak lepas dari penelitian sebelumnya yang telah dilakukan, sehingga penelitian yang akan dilakukan memiliki hubungan antara persamaan dan perbedaan objek yang diteliti. Ringkasan dari penelitian terdahulu dapat dilihat pada Tabel 2.1

Tabel 2. 1 Penelitian Terkait

No	Judul	Metode	Data	Hasil
1	*Detection of Domestic Waste Based on YOLO* (Yaohui Hou, 2022)	*You Only Look Once* (YOLO) v5	Dataset : VOC berisi 44 kelas sampah rumah tangga seperti *snack box*, produk pencuci, plastik, dan lain-lain	*Precision*: 65% *Recall* : 50%
2	*Multi object detection and classification in solid waste management using region proposal network and YOLO model* (Jansi Rani S.V, 2022)	*You Only Look Once* (YOLO) v5	Dataset : terdiri dari 6 kelas, yaitu *glass, cardboard, metal, plastic, paper, trash*	mAP : 0,84
3	*Deep Learning-Based YOLO Models for the Detection of People With Disabilities* (Madallah Alruwaili, 2024)	*You Only Look Once* (YOLO) v5, v7 dan v8	Kumpulan data substansial yang terdiri dari 4.300 gambar dan 8.447 label yang mencakup lima kategori berbeda	YOLO v5 : *Precision* : 0,88 *Recall* : 0,88 YOLO v7 : *Precision* : 0,90

				Recall : 0,92 YOLO v8 : *Precision* : 0,90 *Recall* : 0,94
4	*Fruit ripeness identifcation using YOLOv8 model* (Bingjie Xiao, 2023)	*You Only Look Once* (YOLO) v8	Data terdiri dari empat kelas, yaitu apel matang, apel terlalu matang, pir matang, dan pir terlalu matang	*Accuracy* : 99,5
5	Penerapan Algoritma *You Only Look Once Version 8* Untuk Identifikasi Abjad Bahasa Isyarat Indonesia (Agung Ma'ruf, 2023)	*You Only Look Once* (YOLO) v8	Dataset berjumlah 11.469 setelah melalui proses augmentasi yang terdiri dari 26 kelas.	Akurasi : 99,8% *Presisi* : 99,4 *Recall* : 99,8
6	Deteksi Otomatis Terhadap Pelanggaran Pembuang Sampah Menggunakan	*You Only Look Once* (YOLO) v5	Dataset berjumlah 800 gambar dan dibagi menjadi 2 kelas, yaitu membuang sampah dan tidak membuang sampah	Akurasi : 95%

	Metode *You Only Look Once* (YOLO)			
7	Deteksi Tumpukan Sampah dengan Metode You Only Look Once (YOLO)	*You Only Look Once* (YOLO) v5	Dataset tumpukan sampah diambil menggunakan kamera, Kemudian dibagi sejumlah 100 gambar untuk data latih dan 20 gambar untuk data validasi	*Recall* : 95% *Precision* : 100% F1 *Score* : 97%
8	Sistem Pendeteksi Sampah Secara Realtime Menggunakan Metode YOLO (Kadek Adi Priana, 2023)	*You Only Look Once* (YOLO) v5	Dataset terdiri dari 550 gambar yang dibagi menjadi 12 kategori sampah	*Precision* : 99,8% *Recall* : 100% mAP : 99,5%
9	Implementasi YOLO versi 3 untuk Mengidentifikasi dan Mengklasifikasi Sampah Kantor berbasis	*You Only Look Once* (YOLO) v3	Dataset sebanyak 800 gambar pada setiap kelas, yaitu botol plastik, kaleng dan kertas	Akurasi : 94%

	NVIDIA Jetson Nano			
10	Rancang Bangun Sistem Pengklasifikasi Jenis Sampah Organik dan Anorganik menggunakan metode You Only Look Onceversi 3 berbasis Raspberry Pi (Figo Ramadhan Hendri, 2022)	*You Only Look Once* (YOLO) v3	dataset sampah yang berjumlah 7000 yang terdiri dari dua kelas, yaitu sampah organik dan anorganik	Akurasi : 93,7%

2.13 Skenario Pengujian

Skenario pengujian adalah serangkaian langkah-langkah atau tindakan yang dirancang untuk menguji suatu sistem, aplikasi, atau fitur tertentu dengan tujuan untuk memverifikasi bahwa sistem tersebut berfungsi sesuai dengan persyaratan yang telah ditetapkan [34]. Skenario ini seringkali dibuat berdasarkan kasus penggunaan (*use case*) atau alur pengguna (*user flow*) yang menggambarkan bagaimana pengguna berinteraksi dengan sistem. Pada penelitian ini terdapat sembilan skenario yang akan diuji yang terdapat pada Tabel 2.2 berikut

Tabel 2. 2 Skenario Pengujian

Nama Skenario	Latar Belakang	Pencahayaan	Jarak Objek	Jumlah Objek Deteksi
Skenario 1	Putih polos	Normal (di dalam ruangan yang memiliki pencahayaan dari jendela dan tanpa menyalakan lampu)	30 cm	1 objek
				2 objek
				3 objek
			50 cm	1 objek
				2 objek
				3 objek
			100 cm	1 objek
				2 objek
				3 objek
Skenario 2	Putih polos	Redup (di dalam ruangan tertutup dan menggunakan lampu 5 watt)	30 cm	1 objek
				2 objek
				3 objek
			50 cm	1 objek
				2 objek
				3 objek
			100 cm	1 objek
				2 objek
				3 objek
Skenario 3	Putih polos	Di luar ruangan dengan cahaya matahari	30 cm	1 objek
				2 objek
				3 objek
			50 cm	1 objek
				2 objek
				3 objek
			100 cm	1 objek
				2 objek

				3 objek
Skenario 4	Berwarna dan berpola gambar	Normal (di dalam ruangan yang memiliki pencahayaan dari jendela dan tanpa menyalakan lampu)	30 cm	1 objek
				2 objek
				3 objek
			50 cm	1 objek
				2 objek
				3 objek
			100 cm	1 objek
				2 objek
				3 objek
Skenario 5	Berwarna dan berpola gambar	Redup (di dalam ruangan tertutup dan menggunakan lampu 5 watt)	30 cm	1 objek
				2 objek
				3 objek
			50 cm	1 objek
				2 objek
				3 objek
			100 cm	1 objek
				2 objek
				3 objek
Skenario 6	Berwarna dan berpola gambar	Di luar ruangan dengan cahaya matahari	30 cm	1 objek
				2 objek
				3 objek
			50 cm	1 objek
				2 objek
				3 objek
			100 cm	1 objek
				2 objek
				3 objek
Skenario 7		Normal (di dalam	30 cm	1 objek
				2 objek

	Berbagai objek lain di latar belakang	ruangan yang memiliki pencahayaan dari jendela dan tanpa menyalakan lampu)		3 objek
			50 cm	1 objek
				2 objek
				3 objek
			100 cm	1 objek
				2 objek
				3 objek
Skenario 8	Berbagai objek lain di latar belakang	Redup (di dalam ruangan tertutup dan menggunakan lampu 5 watt)	30 cm	1 objek
				2 objek
				3 objek
			50 cm	1 objek
				2 objek
				3 objek
			100 cm	1 objek
				2 objek
				3 objek
Skenario 9	Berbagai objek lain di latar belakang	Di luar ruangan dengan cahaya matahari	30 cm	1 objek
				2 objek
				3 objek
			50 cm	1 objek
				2 objek
				3 objek
			100 cm	1 objek
				2 objek
				3 objek

Adapun deskripsi dari skenario pengujian pada Tabel 2.2 adalah sebagai berikut

1. Skenario Pertama

   Skenario pengujian ini dilakukan dengan menggunakan latar belakang putih polos sebagai latar belakang utama. Objek yang diuji ditempatkan pada latar belakang ini. Pencahayaan yang digunakan adalah cahaya alami yang masuk melalui jendela tanpa adanya pencahayaan buatan (lampu). Kondisi pencahayaan ini dianggap sebagai kondisi pencahayaan normal di dalam ruangan. Kombinasi latar belakang putih polos dan pencahayaan alami bertujuan untuk meminimalisir pengaruh faktor eksternal terhadap hasil pengujian dan memfokuskan perhatian pada karakteristik objek yang diuji. Objek yang di deteksi ditempatkan pada jarak 30 cm, 50 cm, dan 100 cm dari kamera yang masing masing mendeteksi satu, dua, dan tiga objek.

2. Skenario Kedua

   Skenario pengujian ini dilakukan dengan menempatkan objek yang akan diuji pada latar belakang putih polos di dalam ruangan tertutup. Satu-satunya sumber cahaya adalah lampu 5 watt yang menghasilkan pencahayaan yang sangat redup. Kondisi pencahayaan yang minim ini bertujuan untuk mensimulasikan kondisi cahaya rendah yang sering ditemui dalam kehidupan sehari-hari. Objek yang di deteksi ditempatkan pada jarak 30 cm, 50 cm, dan 100 cm dari kamera yang masing masing mendeteksi satu, dua, dan tiga objek.

3. Skenario Ketiga

   Skenario pengujian ini dilakukan dengan menempatkan objek yang akan diuji pada latar belakang putih polos di luar ruangan. Cahaya matahari menjadi sumber cahaya utama. Kondisi pencahayaan ini bertujuan untuk mensimulasikan kondisi pencahayaan alami di luar ruangan yang sering ditemui dalam kehidupan sehari-hari. Objek yang di deteksi ditempatkan pada jarak 30 cm, 50 cm, dan 100 cm dari kamera yang masing masing mendeteksi satu, dua, dan tiga objek.

4. Skenario Keempat

   Skenario pengujian ini dilakukan dengan menempatkan objek yang akan diuji pada latar belakang yang berwarna dan berpola di dalam ruangan yang mendapat cahaya alami dari jendela. Kombinasi antara latar belakang yang kompleks dan pencahayaan alami akan menciptakan kondisi pengujian yang lebih menantang dibandingkan dengan skenario sebelumnya. Objek yang di deteksi ditempatkan pada jarak 30 cm, 50 cm, dan 100 cm dari kamera yang masing masing mendeteksi satu, dua, dan tiga objek.

5. Skenario Kelima

   Skenario pengujian ini dilakukan dengan menempatkan objek yang akan diuji pada latar belakang yang berwarna dan berpola di dalam ruangan tertutup yang hanya diterangi oleh lampu 5 watt. Kombinasi antara latar belakang yang kompleks dan pencahayaan redup akan menciptakan kondisi pengujian yang sangat menantang. Objek yang di deteksi ditempatkan pada jarak 30 cm, 50 cm, dan 100 cm dari kamera yang masing masing mendeteksi satu, dua, dan tiga objek.

6. Skenario Keenam

   Skenario pengujian ini dilakukan dengan menempatkan objek yang akan diuji pada latar belakang yang berwarna dan berpola di luar ruangan. Cahaya matahari secara langsung menjadi sumber pencahayaan utama. Kombinasi antara latar belakang yang kompleks dan pencahayaan alami yang dinamis menciptakan kondisi pengujian yang sangat menantang. Objek yang di deteksi ditempatkan pada jarak 30 cm, 50 cm, dan 100 cm dari kamera yang masing masing mendeteksi satu, dua, dan tiga objek.

7. Skenario Ketujuh

   Skenario pengujian ini dilakukan dengan menempatkan objek utama yang akan diuji di tengah-tengah berbagai objek lain di dalam ruangan yang mendapat cahaya alami dari jendela. Kombinasi antara latar belakang yang

kompleks dan pencahayaan alami akan menciptakan kondisi pengujian yang lebih menantang dibandingkan dengan skenario sebelumnya. Objek yang di deteksi ditempatkan pada jarak 30 cm, 50 cm, dan 100 cm dari kamera yang masing masing mendeteksi satu, dua, dan tiga objek.

8. Skenario Kedelapan

   Skenario pengujian ini dilakukan dengan menempatkan objek utama yang akan diuji di tengah-tengah berbagai objek lain di dalam ruangan tertutup yang hanya diterangi oleh lampu 5 watt. Kombinasi antara latar belakang yang kompleks dan pencahayaan redup akan menciptakan kondisi pengujian yang sangat menantang. Objek yang di deteksi ditempatkan pada jarak 30 cm, 50 cm, dan 100 cm dari kamera yang masing masing mendeteksi satu, dua, dan tiga objek.

9. Skenario Kesembilan

   Skenario pengujian ini dilakukan dengan menempatkan objek utama yang akan diuji di tengah-tengah berbagai objek lain di luar ruangan. Cahaya matahari secara langsung menjadi sumber pencahayaan utama. Kombinasi antara latar belakang yang kompleks dan pencahayaan alami yang dinamis menciptakan kondisi pengujian yang sangat menantang. Objek yang di deteksi ditempatkan pada jarak 30 cm, 50 cm, dan 100 cm dari kamera yang masing masing mendeteksi satu, dua, dan tiga objek.