

## **BAB II**

### **KAJIAN LITERATUR**

#### **2.1 Keselamatan dan Kesehatan Kerja (K3)**

Keselamatan kerja berfokus pada upaya pencegahan kecelakaan, cedera, dan insiden di tempat kerja, sedangkan kesehatan kerja mencakup upaya menjaga kondisi fisik dan mental tenaga kerja [16]. Peraturan Pemerintah (PP) Nomor 50 Tahun 2012 tentang Penerapan Sistem Manajemen Keselamatan Dan Kesehatan Kerja mendefinisikan K3 sebagai upaya untuk menjamin dan melindungi keselamatan serta kesehatan tenaga kerja melalui pencegahan kecelakaan dan penyakit akibat kerja, sehingga menekankan pendekatan yang sistematis sejak tahap awal.

Secara regulasi, penerapan K3 merupakan kewajiban bagi setiap perusahaan. Undang-Undang Nomor 13 Tahun 2003 dan Undang-Undang Nomor 1 Tahun 1970 menegaskan pentingnya pelaksanaan standar keselamatan kerja untuk mengurangi risiko kecelakaan. Selain itu, Peraturan Menteri Pekerjaan Umum Nomor 09/PRT/M/2008 mengatur penerapan sistem manajemen K3 konstruksi secara menyeluruh, mulai dari perencanaan, pelaksanaan, hingga pengawasan proyek. Penerapannya mencakup identifikasi bahaya, penggunaan APD, pengawasan kepatuhan, peningkatan kompetensi pekerja, serta pemanfaatan teknologi untuk meminimalkan risiko kecelakaan.

Seiring dengan perkembangan teknologi, pemanfaatan kecerdasan buatan menjadi salah satu pendekatan inovatif dalam meningkatkan efektivitas penerapan K3 [17]. Salah satu implementasinya adalah sistem deteksi otomatis penggunaan Alat Pelindung Diri (APD), khususnya helm keselamatan, yang berperan dalam meningkatkan kepatuhan pekerja serta akurasi pengawasan tanpa menambah beban pengawas. Dalam penelitian ini, helm keselamatan diposisikan sebagai APD utama untuk mencegah cedera kepala di lingkungan konstruksi.



(a) Manusia Pakai Helm



(b) Manusia Pakai Helm



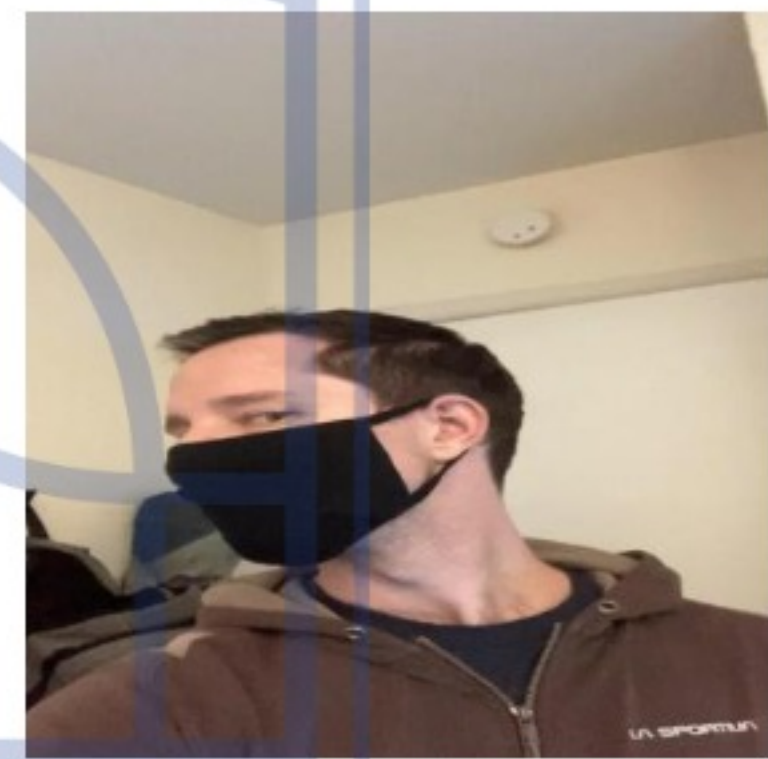
(c) Objek Kecil dan Jauh



(d) Cone



(e) Mesin Industri



(f) Manusia Pakai Helm

Gambar 2. 1 Contoh Data yang Digunakan pada *Dataset*

Sebagai pendukung pengembangan sistem, penelitian ini menggunakan *dataset* yang diperoleh dari [Kaggle](https://www.kaggle.com/) yang memuat variasi kondisi lapangan, seperti pekerja dengan dan tanpa APD, objek berukuran kecil, serta keberadaan peralatan konstruksi. Variasi ini bertujuan meningkatkan kemampuan model dalam mendeteksi objek pada berbagai situasi kerja. Contoh *dataset* ditunjukkan pada Gambar 2.1.

## 2.2 Deteksi Objek

*Object detection* atau deteksi objek adalah proses untuk menentukan keberadaan serta lokasi objek tertentu dalam citra digital [18]. Teknik ini termasuk dalam bidang *Computer vision* dan digunakan untuk mengidentifikasi objek pada gambar maupun video. Secara konseptual, *object detection* menggabungkan dua tugas utama secara bersamaan, yaitu lokalisasi objek (*object localization*) dan klasifikasi objek (*object classification*) [19]. Proses lokalisasi berfungsi menentukan posisi objek melalui koordinat *bounding box*, sedangkan klasifikasi bertujuan mengidentifikasi jenis objek yang terdeteksi.



Gambar 2. 2 Contoh Objek *Occlusion*

Dalam praktiknya, *object detection* menuntut model untuk mampu mengenali lebih dari satu objek dalam satu citra, termasuk pada kondisi ketika objek saling menutupi atau berada pada kualitas visual yang kurang ideal seperti pada gambar 2.2 di atas [17]. Dengan kemampuan mengenali dan menandai objek secara otomatis, *object detection* banyak dimanfaatkan pada sistem pengawasan, kendaraan otonom, deteksi kematangan buah, serta peningkatan keselamatan kerja di proyek konstruksi melalui pemantauan penggunaan Alat Pelindung Diri [17, 20, 21].

### 2.3 Deep Learning dan *Convolutional Neural Network* (CNN)

Dalam beberapa tahun terakhir, teknologi *deep learning* berbasis *Convolutional Neural Network* (CNN) semakin berkembang dan banyak digunakan dalam berbagai tugas pemrosesan citra, seperti pengenalan pola, klasifikasi objek, hingga *object detection* [22]. *Deep learning* memanfaatkan jaringan saraf tiruan berlapis untuk mempelajari representasi data secara bertingkat, mulai dari fitur sederhana hingga pola kompleks [23].

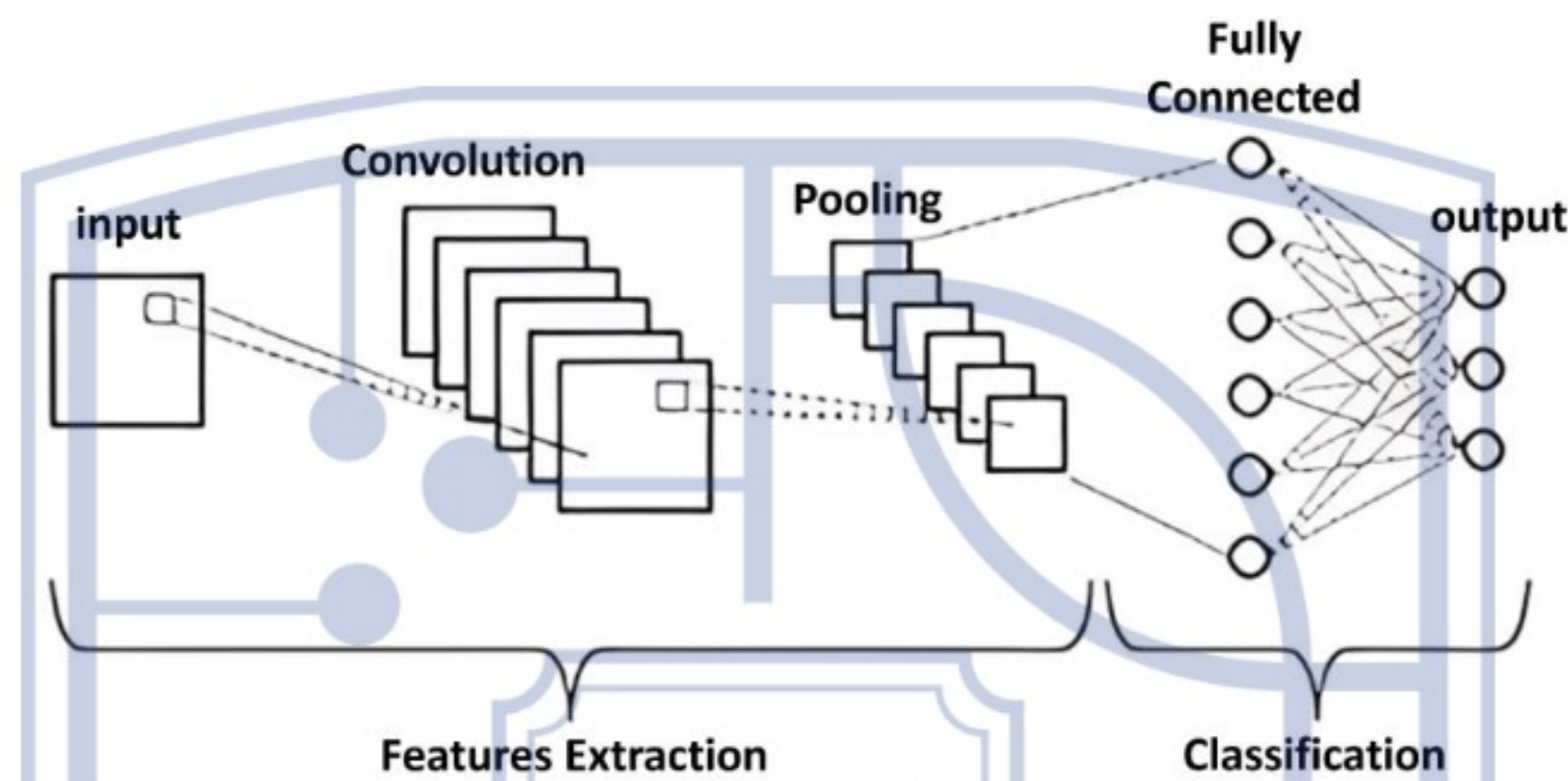
CNN dirancang khusus untuk data citra karena mampu mengekstraksi fitur visual secara otomatis melalui operasi konvolusi. Pada lapisan awal, CNN mengenali pola dasar seperti tepi dan tekstur, sedangkan pada lapisan yang lebih dalam, CNN mampu memahami bentuk objek yang lebih kompleks [24]. Kemampuan ini menjadikan CNN sebagai fondasi utama dalam berbagai metode deteksi objek modern.

Dalam konteks deteksi objek pada lingkungan konstruksi, CNN memainkan peran penting dalam mengenali ciri khas alat pelindung diri (APD) seperti helm keselamatan [25]. CNN juga menjadi fondasi arsitektur berbagai algoritma deteksi terkini seperti YOLO (*You*

*Only Look Once*), yang digunakan dalam penelitian ini untuk melakukan deteksi helm secara cepat dan akurat di lingkungan konstruksi [26, 27].

### 2.3.1 Arsitektur Dasar CNN

Secara umum, lapisan layer CNN dibagi menjadi dua seperti yang di tunjukkan oleh gambar 2.3, yaitu :



Gambar 2. 3 Arsitektur CNN [28]

#### 1. Feature Extraction

Lapisan ini berfungsi untuk mengekstraksi ciri-ciri penting dari citra masukan. Letaknya berada pada bagian awal arsitektur CNN dan tersusun atas beberapa lapisan yang saling berhubungan. Lapisan utama pada bagian ini adalah *convolutional layer*, *Pooling Layer*, dan ReLU [29, 30].

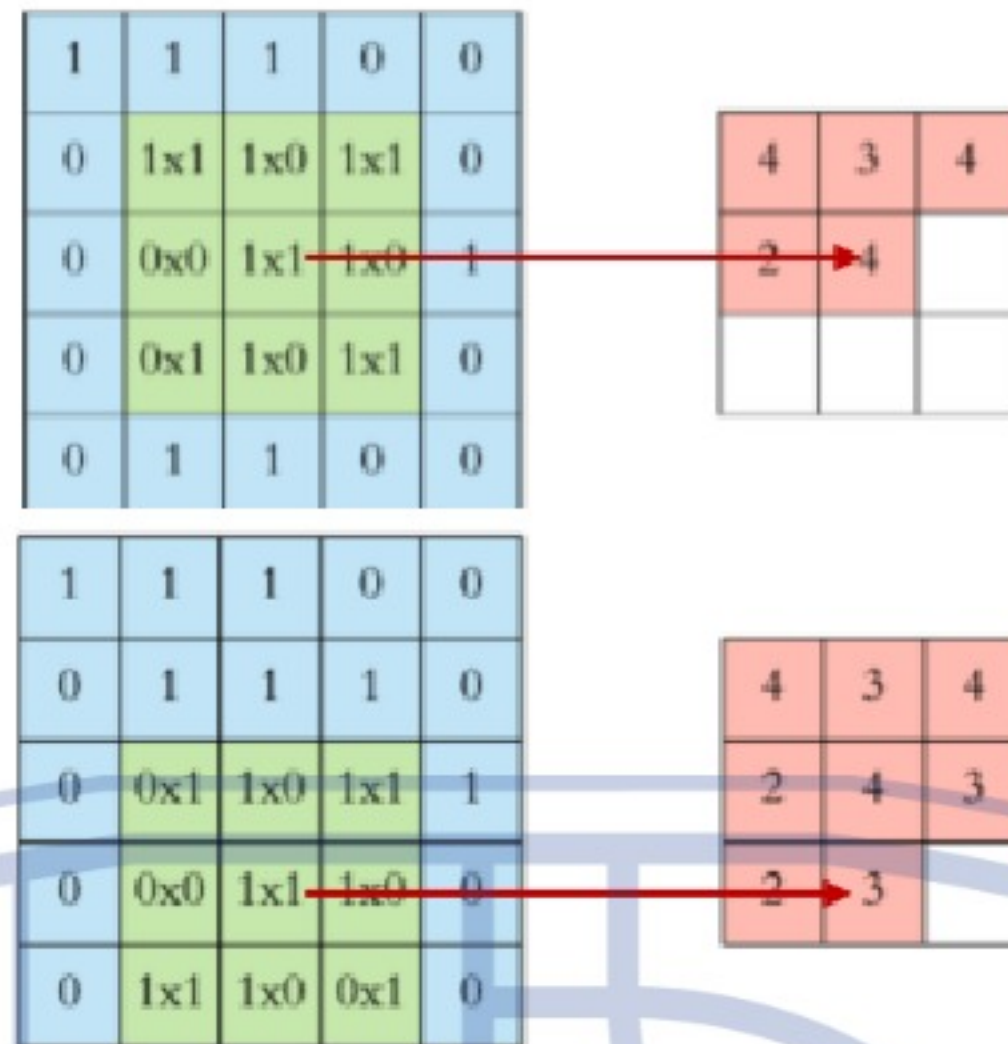
#### 2. Classification Layer

Setelah melalui proses ekstraksi fitur, hasil kemudian diteruskan ke lapisan klasifikasi. Bagian ini terdiri dari beberapa lapisan dengan neuron yang terhubung penuh (*Fully Connected Layer*) [31, 32]. Lapisan klasifikasi bekerja dengan cara menghasilkan vektor dengan dimensi K, yang mana K merupakan jumlah kelas yang dapat diprediksi oleh jaringan [32]. Keluaran dari lapisan ini digunakan untuk menentukan kelas akhir dari citra *input*, misalnya apakah termasuk kategori helm, *vest*, atau objek lain sesuai data latih.

Pada penelitian ini, terdapat tiga macam lapisan utama yang membentuk arsitektur dasar CNN, yaitu:

#### 1. Convolutional layer

Lapisan ini merupakan komponen utama CNN yang berfungsi mengekstraksi fitur dari citra menggunakan filter *kernel* dan *stride*. Filter akan bergerak pada citra untuk menangkap pola seperti tepi, sudut, atau tekstur [31].

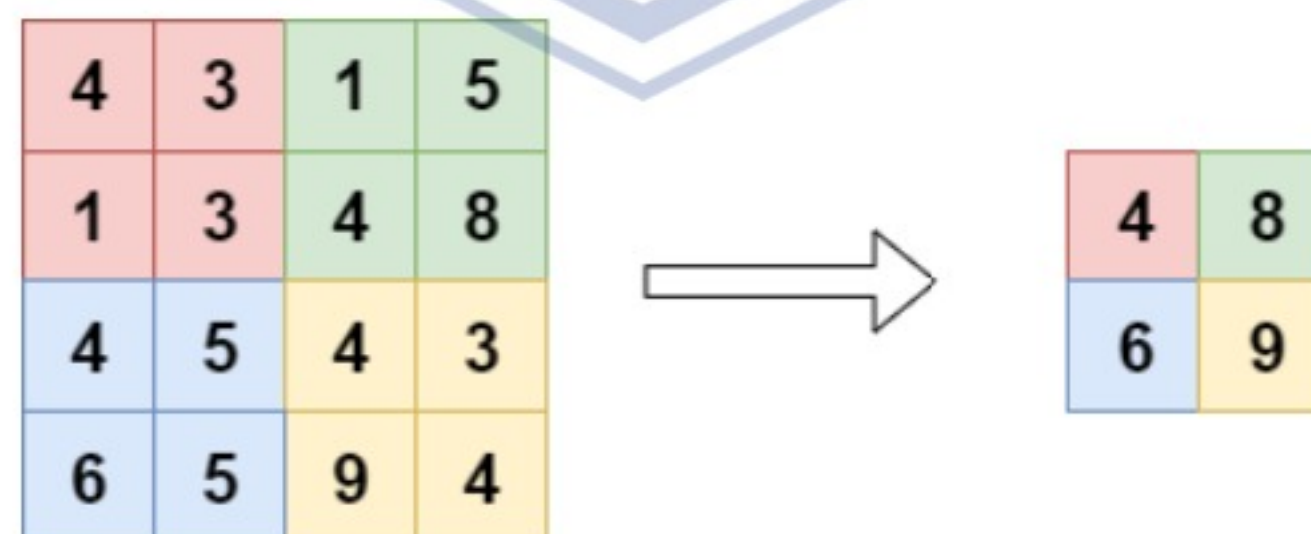


Gambar 2. 4 Convolution Process [33]

Kotak biru pada gambar 2.4 di atas adalah citra yang akan di konvolusi. Sebuah *kernel* atau filter akan bergerak dari sudut kiri atas citra menuju ke kanan bawah, kemudian menghasilkan *feature map* pada *output*. Proses konvolusi bertujuan untuk mengekstraksi fitur penting dari citra, seperti tepi, tekstur, atau pola tertentu, dengan memanfaatkan informasi spasial yang ada pada data [31, 34].

## 2. Pooling Layer

*Pooling layer* berfungsi untuk mereduksi dimensi fitur tanpa menghilangkan informasi penting, sehingga dapat mengurangi kompleksitas komputasi dan membantu mencegah *overfitting* [35]. Jenis *pooling* yang paling umum digunakan adalah *max pooling*, yaitu memilih nilai maksimum dari area tertentu pada *feature map* [29, 31, 36]. Sebagai contoh, ketika digunakan *max pooling* berukuran 2x2 dengan *stride* 2, maka filter akan bergerak melintasi citra dengan langkah dua piksel, dan pada setiap langkahnya hanya nilai terbesar dari area 2x2 yang dipertahankan.



Gambar 2. 5 Proses Max pooling [37]

Seperti ditunjukkan pada Gambar 2.5, *grid* berwarna merah, kuning, hijau, dan biru di sebelah kiri merupakan kelompok piksel sebelum dilakukan pemilihan nilai maksimum.

Setelah melalui proses *pooling*, area tersebut diperkecil menjadi *grid* berukuran 2x2 di sebelah kanan yang hanya berisi nilai terbesar dari masing-masing kelompok.

Dengan cara ini, ciri penting dari gambar tetap dipertahankan meskipun gambar mengalami sedikit pergeseran atau perubahan posisi. Artinya, *max pooling* membantu jaringan tetap bisa mengenali objek secara baik, walaupun posisi atau ukuran objek di dalam gambar tidak selalu sama.

### 3. *Fully Connected Layer*

Lapisan ini berfungsi melakukan klasifikasi berdasarkan fitur yang telah diekstraksi sebelumnya, dengan menghasilkan probabilitas untuk setiap kelas objek [32].

Kombinasi lapisan-lapisan tersebut memungkinkan CNN mempelajari representasi visual secara efektif untuk mendukung proses deteksi objek.

#### 2.3.2 Fungsi CNN dalam Ekstraksi Fitur Citra

Fungsi utama CNN dalam ekstraksi fitur adalah menyaring informasi visual dari tingkat sederhana hingga kompleks. Pada lapisan awal, CNN mendeteksi ciri-ciri dasar seperti tepi, garis, atau sudut [34]. Pada lapisan yang lebih dalam, CNN mampu mengenali pola yang lebih kompleks, misalnya bentuk wajah, kendaraan, atau helm keselamatan.

Keunggulan CNN adalah kemampuannya untuk melakukan *feature learning* secara otomatis, sehingga tidak diperlukan perancangan fitur manual (*hand-crafted features*) seperti pada metode konvensional [38]. Hal ini membuat CNN lebih adaptif terhadap berbagai variasi data, seperti perubahan pencahayaan, posisi, dan ukuran objek.

#### 2.3.3 Penerapan CNN dalam Object Detection

Dalam *computer vision*, CNN digunakan sebagai dasar dalam sistem *object detection*, yaitu untuk mendeteksi keberadaan objek sekaligus menentukan lokasinya melalui *bounding box*. CNN berperan sebagai *feature extractor* yang membantu model mengenali area citra yang mengandung objek.

Perkembangan metode deteksi berbasis CNN telah menghasilkan berbagai algoritma modern, salah satunya adalah YOLO (*You Only Look Once*). Algoritma ini mengintegrasikan proses ekstraksi fitur dan prediksi objek dalam satu tahap, sehingga mampu melakukan deteksi secara cepat dan efisien [39, 40].

## 2.4 YOLO (*You Only Look Once*)

YOLO adalah salah satu algoritma yang dapat mendeteksi objek dengan kecepatan proses dan tingkat akurasi yang tinggi [39]. YOLO lebih mengutamakan efisiensi dengan memproses citra dalam satu kali perhitungan, sehingga mampu mengenali objek dengan cepat dan tetap menjaga kualitas deteksi [41, 42].

Tujuan utama dari pengembangan YOLO adalah menciptakan detektor objek yang dapat bekerja cepat pada sistem produksi, serta dapat dioptimalkan dengan komputasi paralel untuk menghasilkan kinerja yang efisien pada perangkat keras dengan kapasitas terbatas [43]. Selain itu, algoritma ini juga relatif mudah dilatih dan diimplementasikan, bahkan dengan GPU kelas menengah sekalipun tetap dapat menghasilkan deteksi objek yang cukup baik.

### 2.4.1 Perbandingan YOLOv8 Dibanding Algoritma YOLO Lain

YOLOv8 merupakan salah satu versi populer dari keluarga algoritma YOLO (*You Only Look Once*). Meskipun saat ini sudah tersedia versi terbaru yaitu YOLOv11, penggunaan YOLOv8 tetap relevan karena kestabilannya dan juga akurasi dalam mendeteksi objek kecil serta dukungan komunitas yang sangat luas [44]. Beberapa keunggulan utama YOLOv8 dibanding versi YOLO lainnya adalah sebagai berikut:

#### 1. Kecepatan dan Efisiensi

YOLOv8 mampu melakukan deteksi objek dengan latensi rendah, dan YOLOv8 menawarkan keseimbangan yang lebih baik antara akurasi dan kecepatan. Dibandingkan versi lama seperti YOLOv3 dan YOLOv5, YOLOv8 menunjukkan peningkatan signifikan dalam akurasi, efisiensi, dan stabilitas deteksi, sehingga lebih sesuai untuk aplikasi pemantauan keselamatan kerja yang membutuhkan hasil deteksi yang lebih konsisten dan andal [44].

#### 2. Arsitektur Lebih Modern

YOLOv8 menggunakan arsitektur yang lebih efisien pada bagian *backbone* dan *head*, sehingga mampu meningkatkan performa deteksi, khususnya pada objek berukuran kecil. Backbone berfungsi mengekstraksi fitur visual dari citra, Neck kemudian menyempurnakan dan menggabungkan fitur multi-skala yang diperoleh kemudian dioptimalkan menggunakan lapisan seperti *upsample*, *concat*, dan *convolution* untuk meningkatkan deteksi objek dengan berbagai ukuran, sedangkan *head* bertugas

menghasilkan prediksi akhir berupa *bounding box*, kelas objek, dan *confidence score* [45, 46].

### 3. Akurasi Tinggi

YOLOv8 terbukti memberikan akurasi lebih tinggi dibandingkan YOLOv3, YOLOv4, maupun YOLOv5, tanpa mengorbankan kecepatan inferensi [47]. Meskipun YOLOv10 dikenal sebagai salah satu model tercepat untuk mendeteksi, YOLOv8 masih lebih unggul dalam hal stabilitas pelatihan dan dukungan implementasi. Misalnya, pada kasus deteksi objek kendaraan yang bervariasi, YOLOv8 menunjukkan sedikit keunggulan dalam mendeteksi mobil dibandingkan YOLOv10 dan kedua model menunjukkan kemampuan yang setara dalam mendeteksi bus dan sepeda motor [48].

### 4. Kemudahan Implementasi

YOLOv8 dikembangkan resmi oleh Ultralytics, sehingga dokumentasi, API, dan integrasinya lebih sederhana dibandingkan beberapa versi sebelumnya (YOLOv4, YOLOv6, YOLOv7) [49]. Hal ini memudahkan peneliti maupun praktisi dalam proses pelatihan dan *deployment*.

### 5. Optimasi Augmentasi Data

YOLOv8 mendukung berbagai teknik *augmentasi* data seperti *mosaic*, *mixup*, *flipping*, hingga *scaling* [44].

### 6. Dukungan Multi-varian (n, s, m, l, x)

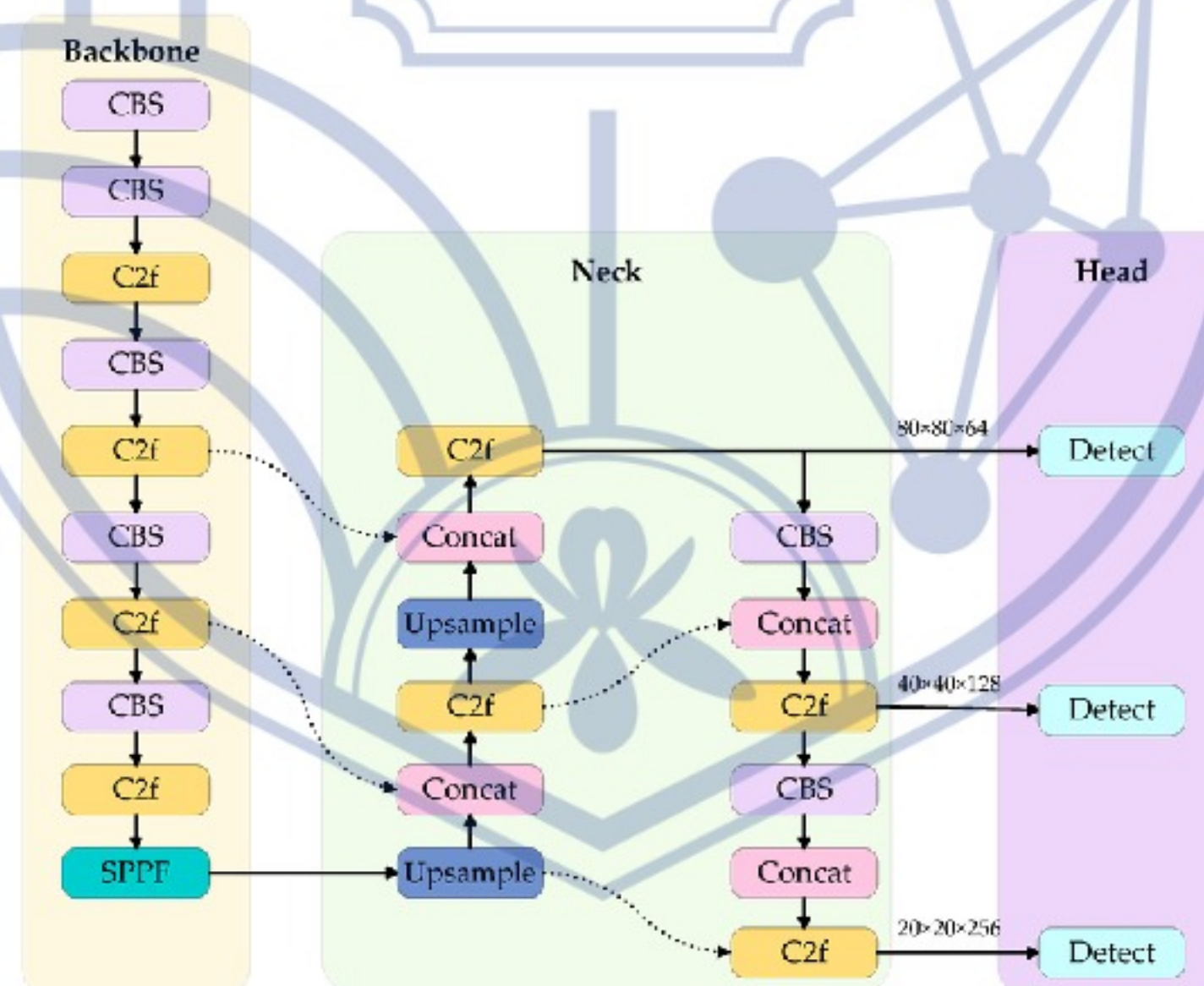
Sama halnya dengan YOLOv5, YOLOv8 hadir dengan varian ukuran yang dapat disesuaikan dengan kebutuhan, mulai dari YOLOv8n (*nano*) untuk efisiensi hingga YOLOv8x (*extra-large*) untuk akurasi maksimal [47, 50].

Secara umum, YOLOv3 dikenal ringan namun kurang akurat untuk objek kecil, sedangkan YOLOv4 dan YOLOv5 meningkatkan akurasi tetapi membutuhkan optimasi tambahan untuk efisiensi [47]. YOLOv7 menawarkan efisiensi komputasi yang baik, namun implementasinya lebih kompleks. Sementara itu, YOLOv10 memiliki kecepatan tinggi tetapi cenderung lebih sensitif terhadap variasi kondisi data [48]. Sementara itu, YOLOv11 merupakan versi terbaru dengan peningkatan performa lebih lanjut, namun masih relatif baru sehingga dokumentasi serta dukungan ekosistemnya belum selengkap YOLOv8.

Pemilihan ini juga diperkuat oleh sejumlah penelitian terbaru yang secara khusus menerapkan YOLOv8 pada kasus deteksi helm keselamatan di lingkungan kerja yang kompleks. Penelitian yang membandingkan performa YOLOv3, YOLOv4, YOLOv5, YOLOv7, YOLOv8, YOLOv10, dan YOLOv11 pada tugas deteksi helm keselamatan menunjukkan bahwa YOLOv8 tetap menjadi basis pengembangan yang unggul untuk menangani tantangan seperti helm berukuran kecil dan latar belakang yang kompleks [15]. Selain itu, studi lain yang membandingkan YOLOv8 dengan algoritma lain seperti Faster R-CNN, SSD, YOLOv9, dan YOLO11 menemukan bahwa YOLOv8 memberikan keseimbangan yang lebih baik antara akurasi deteksi dan kompleksitas model dibandingkan algoritma tersebut [12], sehingga menguatkan alasan pemilihan YOLOv8 sebagai basis model pada penelitian ini meskipun versi yang lebih baru telah tersedia.

#### 2.4.2 Arsitektur YOLOv8 dan Varian (n, s, m, l, x)

Arsitektur YOLOv8 terdiri atas tiga komponen utama, yaitu *Backbone*, *Neck*, dan *Head*, yang bekerja secara terintegrasi untuk melakukan ekstraksi fitur, penggabungan informasi, serta prediksi akhir pada citra. Untuk arsitektur dari YOLOv8 ini dapat di lihat pada gambar 2.6 di bawah :



Gambar 2. 6 Tiga Komponen Utama Arsitektur YOLOv8 [51]

##### 1. *Backbone*

Backbone berfungsi mengekstraksi fitur dari citra *input* dan menghasilkan *feature map* sebagai representasi visual [52]. Proses ini memungkinkan model mengenali pola dasar hingga fitur kompleks yang menjadi dasar dalam membedakan objek, terutama pada kondisi objek saling berdekatan atau mengalami *occlusion*. Dengan demikian, *Backbone* yang

umumnya berupa jaringan *Convolutional Neural Network* (CNN) bertingkat mengambil gambar *input* dan menghasilkan peta fitur (*feature maps*) yang menjadi dasar bagi proses deteksi berikutnya [52].

## 2. Neck

*Neck* berfungsi menggabungkan fitur dari berbagai tingkat resolusi agar informasi yang diperoleh lebih lengkap [53]. Sebagai contoh, ketika mendeteksi manusia, *Neck* akan menggabungkan informasi detail kecil seperti bentuk helm dengan informasi skala lebih besar.

## 3. Head

*Head* adalah bagian akhir yang menghasilkan prediksi berupa *bounding box*, *confidence score*, serta kelas objek [52]. Misalnya, setelah melalui *Backbone* dan *Neck*, *Head* akan menentukan bahwa area tertentu pada citra adalah *helm* dengan tingkat keyakinan 90%, atau mendeteksi *orang tanpa helm* dengan tingkat keyakinan tertentu. Bagian ini menjadi keluaran utama yang digunakan dalam sistem deteksi objek.

Dapat dilihat pada Gambar 2.6, arsitektur YOLOv8 terdiri atas tiga bagian utama, yaitu *Backbone*, *Neck*, dan *Head*. Pada bagian *Backbone* digunakan modul CBS (*Convolution, Batch Normalization, SiLU*) yang berfungsi mengekstraksi fitur dari citra masukan melalui lapisan konvolusi, normalisasi batch, dan fungsi aktivasi SiLU. Selain itu, digunakan modul C2f yang merupakan versi optimal dari modul C2 (*Cross Stage Partial Bottleneck with Two Convolutions*). Modul ini memanfaatkan *residual connection* untuk meningkatkan representasi fitur dan memperbaiki propagasi gradien sehingga proses pelatihan menjadi lebih efisien tanpa mengurangi performa model. *Backbone* juga dilengkapi dengan modul SPPF (*Spatial Pyramid Pooling Fast*) yang berfungsi menggabungkan fitur pada berbagai tingkat melalui proses *pooling*, sehingga model mampu menangkap informasi yang lebih komprehensif dari objek dengan ukuran yang bervariasi [54].

Setelah proses ekstraksi fitur pada *Backbone*, bagian *Neck* bertugas menggabungkan fitur dari berbagai skala menggunakan teknik *feature aggregation* dan *concatenation*. Proses ini memungkinkan model memanfaatkan informasi dari berbagai tingkat resolusi sehingga representasi fitur menjadi lebih kaya dan kemampuan deteksi objek pada berbagai ukuran dapat ditingkatkan [54]. Selanjutnya, bagian *Head* menghasilkan prediksi akhir berupa

lokasi *bounding box*, skor kepercayaan (*objectness*), dan probabilitas kelas untuk setiap objek yang terdeteksi [54].

YOLOv8 tersedia dalam lima varian, yaitu n (*nano*), s (*small*), m (*medium*), l (*large*), dan x (*extra-large*) yang dibedakan berdasarkan jumlah parameter dan kompleksitas arsitekturnya. Perbedaan ini memengaruhi keseimbangan antara kecepatan inferensi, kebutuhan komputasi, dan tingkat akurasi. Pada penelitian TURJAF (*Turkish Journal of Agriculture -Food Science and Technology*) yang menjadi rujukan, seluruh varian YOLOv8 diuji menggunakan *dataset* berisi 7 kelas dengan total 3500 citra [55]. Setiap varian dilatih dengan pengaturan dan kondisi eksperimen yang sama, sehingga perbandingan performanya dapat dinilai secara objektif berdasarkan perbedaan kompleksitas model, kebutuhan komputasi, serta akurasinya sebagaimana ditampilkan pada Tabel 2.1 dibawah.

Tabel 2. 1 Perbandingan performa dari varian YOLOv8 [55]

| Model   | Parameters (M) | Layers | Training Time (days) | GFLOPs | GPU Usage (GB) | mAP 50 | mAP50-95 | Precision (P) | Recall (R) |
|---------|----------------|--------|----------------------|--------|----------------|--------|----------|---------------|------------|
| YOLOv8n | 3              | 168    | 17                   | 8.1    | 6.3            | 0.909  | 0.629    | 0.881         | 0.863      |
| YOLOv8s | 11             | 168    | 18                   | 28.5   | 7.7            | 0.907  | 0.625    | 0.878         | 0.858      |
| YOLOv8m | 25             | 218    | 22                   | 78.7   | 10.5           | 0.922  | 0.645    | 0.864         | 0.895      |
| YOLOv8l | 43             | 268    | 28                   | 164.8  | 14.5           | 0.916  | 0.635    | 0.883         | 0.890      |
| YOLOv8x | 68             | 268    | 34                   | 257.4  | 17.7           | 0.926  | 0.648    | 0.892         | 0.903      |

#### 1. YOLOv8n (*nano*)

YOLOv8n merupakan varian paling ringan dengan 3 juta parameter dan 8,1 GFLOPs. Konsumsi memori GPU sangat rendah, yaitu 6,3 GB, dan menjadi model dengan waktu pelatihan tercepat. Pada pengujian menggunakan *dataset* 7 kelas (3500 citra),

akurasi mencapai  $mAP50 = 0,909$ ,  $mAP50-95 = 0,629$ ,  $precision = 0,881$ , dan  $recall = 0,863$  sebagaimana ditampilkan pada Tabel 2.1.

## 2. YOLOv8s (*small*)

Varian YOLOv8s memiliki 11 juta parameter dengan 168 layer dan kompleksitas 28,5 GFLOPs. Kebutuhan memori GPU mencapai 7,7 GB, dengan waktu pelatihan sekitar 18 hari. Akurasi yang diperoleh adalah  $mAP50$  sebesar 0,907 dan  $mAP50-95$  sebesar 0,625,  $precision = 0,878$ , dan  $recall = 0,858$ .

## 3. YOLOv8m (*medium*)

YOLOv8m berisi sekitar 25 juta parameter dengan 218 layer dan kompleksitas komputasi sebesar 78,7 GFLOPs. Model ini menggunakan memori GPU sebesar 10,5 GB dengan waktu pelatihan  $\pm 22$  hari. Akurasi lebih tinggi, yaitu  $mAP50$  sebesar 0,922 dan  $mAP50-95$  sebesar 0,645, dengan  $precision = 0,864$  dan  $recall = 0,895$ .

## 4. YOLOv8l (*large*)

Varian ini memiliki 43 juta parameter dan 268 layer, dengan kebutuhan komputasi 164,8 GFLOPs. Konsumsi memori GPU mencapai 14,5 GB dan waktu pelatihan  $\pm 28$  hari. Akurasi yang diperoleh adalah  $mAP50$  sebesar 0,916 dan  $mAP50-95$  sebesar 0,635, dengan  $precision = 0,883$  dan  $recall = 0,890$ .

## 5. YOLOv8x (*extra-large*)

YOLOv8x adalah varian terbesar dengan 68 juta parameter dan 268 layer, serta kompleksitas 257,4 GFLOPs. Model ini membutuhkan 17,7 GB GPU dan waktu pelatihan paling lama, sekitar 34 hari. Tingkat akurasi yang dihasilkan adalah  $mAP50$  sebesar 0,926 dan  $mAP50-95$  sebesar 0,648, dengan  $precision = 0,892$  dan  $recall = 0,903$ .

### 2.4.3 Performa Varian YOLOv8

Hasil evaluasi menunjukkan bahwa setiap varian YOLOv8 memiliki karakteristik yang berbeda dalam hal akurasi dan efisiensi komputasi. Varian kecil seperti YOLOv8n memiliki kecepatan tinggi dan kebutuhan komputasi rendah, namun akurasi relatif lebih rendah [56]. Sebaliknya, varian besar seperti YOLOv8m dan YOLOv8l memiliki jumlah parameter lebih banyak dan membutuhkan sumber daya komputasi yang tinggi, tetapi peningkatan performa yang dihasilkan tidak selalu signifikan, khususnya pada metrik

*precision*. Temuan ini sejalan dengan penelitian lain yang menunjukkan bahwa YOLOv8s mampu melampaui YOLOv8l dengan peningkatan signifikan pada *precision*, kualitas deteksi, *mAP* di berbagai ambang IoU, serta kecepatan inferensi dan FPS, sehingga menawarkan *trade-off* akurasi dan kecepatan yang lebih baik [57].

Berdasarkan hasil evaluasi, YOLOv8s menunjukkan keseimbangan terbaik antara akurasi dan efisiensi. Model ini mampu mencapai nilai *recall*, *precision*, dan *mAP* yang tinggi dengan kebutuhan komputasi yang relatif lebih rendah dibandingkan varian yang lebih besar [57].

Dengan mempertimbangkan kebutuhan sistem dalam mendeteksi penggunaan helm keselamatan pada video, serta adanya tantangan seperti *occlusion* dan kepadatan objek, YOLOv8s dipilih sebagai model yang sesuai dalam penelitian ini. Model ini dinilai mampu memberikan keseimbangan antara akurasi deteksi, kecepatan inferensi, dan efisiensi komputasi, sehingga dapat diintegrasikan dengan algoritma ByteTrack untuk mempertahankan konsistensi identitas objek selama proses pelacakan. Pemilihan varian *small* (YOLOv8s) ini juga selaras dengan sejumlah penelitian deteksi helm keselamatan pada lingkungan konstruksi yang menunjukkan bahwa varian ringan mampu mempertahankan kecepatan deteksi hingga di atas 30 frame per detik, sekaligus tetap efisien dari segi jumlah parameter dan beban komputasi (GFLOPs), sehingga sesuai untuk kebutuhan pemantauan video [58].

#### 2.4.4 Rumus Arsitektur YOLOv8

Arsitektur YOLOv8 terdiri atas beberapa komponen utama yang saling terintegrasi dalam proses ekstraksi fitur, penggabungan fitur, dan deteksi objek. Setiap komponen memiliki fungsi yang berbeda, mulai dari proses konvolusi, normalisasi, aktivasi, hingga pembentukan prediksi objek. Untuk memberikan pemahaman mengenai mekanisme kerja YOLOv8, berikut disajikan rumus-rumus matematis yang merepresentasikan operasi pada setiap komponen penyusun arsitektur YOLOv8.

##### 1. Convolution (Conv)

$$\text{Conv}(x) = W * x + b \dots\dots\dots(1)$$

Keterangan:

- a.  $W = \text{Kernel}$
- b.  $x = \text{Input feature map}$
- c.  $b = \text{Bias}$  (bias yang digunakan pada penelitian ini adalah 0)

## 2. Mean ( $\mu$ )

$$\text{Mean } (\mu) = \mu = \frac{1}{m} \sum_{i=1}^m x_i \dots\dots\dots(2)$$

Keterangan:

- a.  $x$ : *Input feature map*
- b.  $\mu$ : Rata-rata (*mean*) dari *batch*
- c.  $m$ : Jumlah data dalam *batch*

## 3. Variance ( $\sigma^2$ )

$$\text{Variance } (\sigma^2) = \sigma^2 = \frac{1}{m} \sum_{i=1}^m (x_i - \mu)^2 \dots\dots\dots(3)$$

Keterangan:

- a.  $m$ : Jumlah data dalam *batch*
- b.  $\sigma^2$ : *Variance* dari *batch*
- c.  $x$ : *Input feature map*
- d.  $\mu$ : Rata-rata (*mean*) dari *batch*

## 4. Batch Normalization

$$\text{Batch Normalization} = \hat{x} = \frac{x - \mu}{\sqrt{\sigma^2 + \epsilon}} \dots\dots\dots(4)$$

Kemudian diterapkan parameter *learnable*:  $y = \gamma \hat{x} + \beta$

Keterangan:

- a.  $\hat{x}$ : Hasil normalisasi
- b.  $x$ : *Input feature map*
- c.  $\mu$ : Rata-rata (*mean*) dari *batch*
- d.  $\sigma^2$ : *Variance* dari *batch*
- e.  $\epsilon$ : Nilai kecil untuk menghindari pembagian nol (biasanya  $10^{-5}$ )
- f.  $\gamma$ : Parameter skala (*default* = 1)
- g.  $\beta$ : Parameter *shift* (*default* = 0)
- h.  $y$ : *Output* akhir Batch Normalization

## 5. Fungsi Aktivasi SiLU (*Sigmoid Linear Unit*)

$$\text{SiLU}(x) = x * \sigma(x) \dots\dots\dots(5)$$

Keterangan:

- a.  $x$  = Hasil dari *Conv* + Normalisasi
- b.  $\sigma$  = Fungsi sigmoid yang didefinisikan sebagai :

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

c.  $e$  = Bilangan Euler ( $\approx 2.71828$ )

#### 6. C2f (Cross Stage Partial with Two Convolutions)

$$Y_{C2F} = \text{Conv}(\text{Concat}(X, B_1, B_2, \dots, B_n)) \dots \dots \dots (6)$$

Keterangan:

- a.  $X$  = *shortcut feature map*
- b.  $B_1, B_2, \dots, B_n$  = *output beberapa bottleneck*
- c. *Concat* = penggabungan *feature map* antar *channel*
- d. *Conv* = *convolution* akhir
- e.  $Y_{C2F}$  = *output modul C2F*

#### 7. Bottleneck

$$Y_{Bottle} = X + F(X) \dots \dots \dots (7)$$

Keterangan:

- a.  $X$  = *input feature map*
- b.  $F(X)$  = hasil *convolution* dan aktivasi
- c.  $Y_{Bottle}$  = *output bottleneck*

#### 8. Concat

$$F_{concat} = \text{Concat}(X, Y) \dots \dots \dots (8)$$

Keterangan:

- a.  $X$  = *input feature map* pertama
- b.  $Y$  = *input feature map* kedua
- c.  $F_{concat}$  = hasil penggabungan *feature map*

#### 9. Max Pooling

$$Y_{i,j} = \max(X_{m,n}) \dots \dots \dots (9)$$

Keterangan:

- a.  $Y_{i,j}$  = *output pooling*
- b.  $X_{m,n}$  = nilai pada area *kernel pooling*
- c. *max* = nilai maksimum pada area *pooling*

#### 10. SPPF (*Spatial Pyramid Pooling-Fast*)

$$Y_{SPPF} = \text{Concat}(P_1, P_2, P_3, P_4) \dots\dots\dots(10)$$

Keterangan:

- a.  $P_1$  = *feature map* asli
- b.  $P_2$  = hasil *max pooling* pertama
- c.  $P_3$  = hasil *max pooling* kedua
- d.  $P_4$  = hasil *max pooling* ketiga
- e.  $Y_{SPPF}$  = *output SPPF*

#### 11. Upsample

$$Y_{up} = \text{Upsample}(X, s) \dots\dots\dots(11)$$

Keterangan:

- a.  $X$  = *feature map input*
- b.  $s$  = skala pembesaran
- c.  $Y_{up}$  = *output upsample*

### 2.5 Permasalahan *Occlusion* pada Deteksi Objek

*Occlusion* merupakan kondisi di mana suatu objek tertutup sebagian atau seluruhnya oleh objek lain, sehingga menyulitkan proses deteksi dan pengenalan objek secara akurat [59]. Pada lingkungan konstruksi, kondisi ini sering terjadi karena pekerja berada dalam jarak yang berdekatan, saling berinteraksi, serta bergerak secara dinamis.

Dalam konteks *object detection*, *occlusion* dapat menyebabkan model gagal mendeteksi objek atau menghasilkan *bounding box* yang tidak akurat [59]. Hal ini terjadi karena sebagian fitur visual objek tidak terlihat secara utuh, sehingga informasi yang diterima model menjadi tidak lengkap. Selain itu, *occlusion* juga dapat meningkatkan kemungkinan terjadinya kesalahan klasifikasi, terutama pada objek berukuran kecil seperti helm keselamatan [59].

Permasalahan ini menjadi semakin kompleks ketika digunakan pada sistem berbasis video, karena objek yang sebelumnya terdeteksi dapat hilang sementara (*lost track*) akibat tertutup oleh objek lain. Oleh karena itu, diperlukan pendekatan tambahan seperti pelacakan objek untuk menjaga konsistensi identitas objek antar *Frame*

## 2.6 ByteTrack

ByteTrack merupakan algoritma *multi-object tracking* yang melakukan asosiasi terhadap hampir seluruh hasil deteksi, termasuk objek dengan *confidence score* rendah [7, 60]. Algoritma ini memanfaatkan hasil deteksi dengan *confidence* tinggi maupun rendah untuk mempertahankan konsistensi identitas objek antar *Frame* [7]. Pendekatan tersebut membuat ByteTrack lebih stabil dalam menangani kondisi *occlusion*, perubahan pencahayaan, dan objek berukuran kecil sehingga dapat mengurangi *lost track* dan *identity switching* [7, 60].

### 2.6.1 Alasan Pemilihan ByteTrack

Tabel 2.2 menunjukkan perbandingan performa ByteTrack dengan SORT dan DeepSORT berdasarkan beberapa metrik evaluasi pelacakan.

Tabel 2. 2 Skor Performa untuk SORT, DeepSort dan Bytracks [7]

| Metrik           | SORT    | DeepSORT | ByteTrack |
|------------------|---------|----------|-----------|
| MOTA             | 54.7%   | 61.4%    | 77.3%     |
| MOTP             | 77.5%   | 79.1%    | 82.6%     |
| ID Switches      | 831     | 781      | 558       |
| MT               | 34.2%   | 45.1%    | 54.7%     |
| ML               | 24.6%   | 21.3%    | 14.9%     |
| FP               | 7.876   | 5.604    | 3.828     |
| FN               | 26.452  | 21.796   | 14.661    |
| Processing Speed | 143 FPS | 61 FPS   | 171 FPS   |

Berdasarkan Tabel 2.2, ByteTrack memiliki performa pelacakan yang lebih baik dibandingkan SORT dan DeepSORT [7]. Nilai *Multiple Object Tracking Accuracy* (MOTA) sebesar 77,3% menunjukkan kemampuan pelacakan yang lebih akurat, sedangkan nilai *ID Switches* dan *Mostly Lost* (ML) yang lebih rendah menunjukkan konsistensi identitas objek yang lebih baik. Selain itu, ByteTrack juga memiliki kecepatan pemrosesan yang tinggi, yaitu mencapai 171 FPS, yang menunjukkan efisiensi komputasi yang baik tanpa mengurangi kualitas pelacakan [7]. Oleh karena itu, ByteTrack dipilih dalam penelitian ini karena mampu mempertahankan konsistensi identitas objek dengan performa pelacakan yang baik pada kondisi pergerakan dinamis dan *occlusion*.

Dibandingkan dengan BoTSORT, ByteTrack memiliki kualitas pelacakan yang relatif serupa namun dengan kompleksitas komputasi yang lebih rendah [9]. Hal ini membuat

ByteTrack lebih efisien untuk diimplementasikan pada sistem pengawasan berbasis CCTV maupun perangkat *edge* dengan sumber daya terbatas [9].

Berdasarkan keunggulan tersebut, ByteTrack dipilih karena mampu memberikan keseimbangan antara akurasi pelacakan, konsistensi identitas objek, dan efisiensi komputasi. Dengan karakteristik tersebut, ByteTrack sesuai untuk diintegrasikan dengan YOLOv8 dalam mendukung sistem deteksi helm keselamatan pada lingkungan konstruksi yang dinamis.

### 2.6.2 Rumus ByteTrack

ByteTrack melakukan proses pelacakan objek dengan memanfaatkan algoritma Kalman Filter untuk memprediksi posisi objek pada frame berikutnya serta metode *data association* untuk mencocokkan hasil deteksi antar *frame*. Melalui mekanisme tersebut, ByteTrack mampu mempertahankan identitas objek secara konsisten meskipun terjadi pergerakan maupun *occlusion*. Beberapa rumus yang digunakan dalam proses pelacakan ByteTrack dijelaskan sebagai berikut:

#### 1. State Vector Kalman Filter

$$x = [x, y, w, h, v_x, v_y]^T \dots\dots\dots(12)$$

Keterangan:

- a.  $x, y$  = Posisi objek
- b.  $w, h$  = Ukuran *bounding box*
- c.  $v_x, v_y$  = Kecepatan objek

#### 2. Prediksi Kalman Filter

$$x_k^- = Ax_{k-1} \dots\dots\dots(13)$$

Keterangan:

- a.  $x_k^-$  = *State* hasil prediksi
- b.  $A$  = Matriks transisi
- c.  $X_{k-1}$  = *State* sebelumnya

#### 3. Intersection Area

$$a. \quad x_{1,\cap} = \max(x_{1,A}, x_{1,B}) \dots\dots\dots(14)$$

$$b. \quad y_{1,\cap} = \max(y_{1,A}, y_{1,B}) \dots\dots\dots(15)$$

$$c. \quad x_{2,\cap} = \min(x_{2,A}, x_{2,B}) \dots\dots\dots (16)$$

$$d. \quad y_{2,\cap} = \min(y_{2,A}, y_{2,B}) \dots\dots\dots (17)$$

Keterangan:

A, B = *Bounding box* yang dibandingkan

4. Luas Irisan

$$|A \cap B| = (x_{2,\cap} - x_{1,\cap})(y_{2,\cap} - y_{1,\cap}) \dots\dots\dots (18)$$

Keterangan:

$|A \cap B|$  = Luas area irisan

5. Luas *Bounding Box*

$$|A| = (x_{2,A} - x_{1,A})(y_{2,A} - y_{1,A}) \dots\dots\dots (19)$$

$$|B| = (x_{2,B} - x_{1,B})(y_{2,B} - y_{1,B}) \dots\dots\dots (20)$$

Keterangan:

a.  $|A|$  = Luas *bounding box* pertama

b.  $|B|$  = Luas *bounding box* kedua

6. *Intersection over Union* (IoU)

$$IoU(A, B) = \frac{|A \cap B|}{|A| + |B| - |A \cap B|} \dots\dots\dots (21)$$

Keterangan:

a.  $IoU(A, B)$  = Nilai kesesuaian *bounding box*

b.  $|A \cap B|$  = Luas area irisan

c.  $|A|$  = Luas *bounding box* pertama

d.  $|B|$  = Luas *bounding box* kedua

7. *Cost Matrix*

$$C_{ij} = 1 - IoU_{ij} \dots\dots\dots (22)$$

Keterangan:

a.  $C_{ij}$  = Nilai *cost*

b.  $IoU_{ij}$  = Nilai IoU

## 8. Perhitungan *Velocity*

$$v_x = x_t - x_{t-1} \dots\dots\dots(23)$$

$$v_y = y_t - y_{t-1} \dots\dots\dots(24)$$

Keterangan:

- a.  $v_x, v_y$  = Kecepatan objek
- b.  $x_t, y_t$  = Posisi saat ini
- c.  $x_{t-1}, y_{t-1}$  = Posisi sebelumnya

## 9. Prediksi Posisi

$$x_t = x_{t-1} + v_x \dots\dots\dots(25)$$

$$y_t = y_{t-1} + v_y \dots\dots\dots(26)$$

Keterangan:

- a.  $x_t, y_t$  = Posisi hasil prediksi
- b.  $v_x, v_y$  = Kecepatan objek

## 2.7 *Dataset dalam Object detection*

Dalam dunia *object detection*, terdapat beberapa format anotasi yang umum digunakan untuk mendefinisikan lokasi dan label objek dalam citra, antara lain YOLO format, COCO format, dan Pascal VOC format [61]. Pada penelitian ini, format anotasi yang digunakan difokuskan pada YOLO format karena sesuai dengan arsitektur YOLOv8 yang digunakan.

### 1. YOLO Format

Format YOLO menggunakan satu berkas anotasi berekstensi .txt untuk setiap citra. Setiap baris pada berkas tersebut memuat *class id* dan koordinat *bounding box* dengan format sebagai berikut [61]:

$$\left( \frac{x_{center}}{image\ width}, \frac{y_{center}}{image\ height}, \frac{box\ width}{image\ width}, \frac{height}{image\ height} \right)$$

Seluruh koordinat dinyatakan dalam bentuk nilai yang sudah di normalisasi terhadap ukuran citra, sehingga apabila dimensi gambar berubah, nilai anotasi tidak perlu disesuaikan kembali. Setiap citra memiliki satu file anotasi, sedangkan pemetaan *class id* ke nama kelas disimpan pada file terpisah. Format ini digunakan sebagai standar anotasi pada model YOLO selama proses pelatihan [61].

## 2. COCO Format

Format COCO menyimpan seluruh anotasi dataset dalam satu file JSON. Informasi mengenai kelas objek disimpan pada bagian *categories* dan diidentifikasi menggunakan *id*, sedangkan informasi setiap gambar disimpan pada bagian *images*. Data anotasi objek beserta koordinat *bounding box* disimpan pada bagian *annotations*, dengan koordinat titik kiri atas (x, y) serta nilai *width* dan *height* yang dinyatakan secara absolut [61].

## 3. Pascal VOC Format

Format Pascal VOC menggunakan satu file XML untuk setiap citra. Setiap file dapat memuat satu atau lebih *bounding box* beserta label kelas objek. Koordinat *bounding box* dinyatakan menggunakan posisi pojok kiri atas dan pojok kanan bawah secara absolut. Selain informasi kelas objek, format ini juga dapat menyimpan informasi tambahan, misalnya apakah objek terpotong oleh batas gambar (*truncated*) atau mengalami *occlusion* sebagian [61].

Berdasarkan karakteristik tersebut, penelitian ini menggunakan YOLO format sebagai standar anotasi *dataset* karena lebih ringan, sederhana, dan kompatibel dengan model YOLOv8 yang digunakan dalam proses deteksi objek.