# BAB I
# PENDAHULUAN

## 1.1 Latar Belakang

Telur ayam merupakan sumber pangan yang kaya protein berkualitas tinggi, mengandung berbagai asam amino esensial, dan memiliki nilai biologis yang tinggi (Puglisi & Fernandez, 2022). Perkembangan Industri 4.0 mendorong sektor peternakan unggas bertransformasi menuju sistem modern berbasis teknologi digital. Dalam konteks tersebut, kualitas telur menjadi faktor penting yang menentukan nilai jual dan kepuasan konsumen, sehingga perlu diidentifikasi secara cepat dan akurat. Namun, proses pemilahan telur secara manual masih banyak digunakan dan dinilai lambat serta kurang konsisten. Oleh karena itu, teknologi *computer vision* mulai diterapkan untuk mendeteksi dan mengklasifikasikan keretakan telur ayam diatas sistem konveyor industri, karena terbukti lebih cepat, akurat, dan efisien (Akkoyun et al., 2022), (Basri et al., 2025).

Kualitas telur umumnya dinilai berdasarkan kondisi kulit, bentuk, ukuran, dan beratnya. Selain itu, metode *candling* juga digunakan dengan cara menyinari telur diruang gelap untuk memeriksa kondisi bagian dalam telur (Yao et al., 2022). Dalam hal ini, visi komputer berperan penting karena merupakan bidang yang mempelajari bagaimana komputer dapat memahami gambar dan video digital layaknya sistem penglihatan manusia (Mahadevkar et al., 2022). Perkembangan teknologi visi komputer sangat pesat dan telah memainkan peran penting diberbagai bidang seperti otomasi industri, robotika cerdas, dan kendaraan otonom (Gionfrida et al., 2024). Selain itu, dalam robotika dan otomasi industri, visi komputer digunakan untuk persepsi robot terhadap lingkungan, memungkinkan kontrol robot cerdas dan adaptasi terhadap kondisi dinamis (Gionfrida et al., 2024).

Keberhasilan model visi komputer sangat dipengaruhi oleh kemampuan dalam mengekstraksi fitur yang relevan seperti tekstur, bentuk, dan intensitas cahaya agar dapat membedakan objek dari latar belakang (Wu et al., 2024). Deteksi keretakan telur ayam, pola retakan halus, variasi tekstur cangkang, serta kondisi pencahayaan sangat memengaruhi hasil deteksi. Oleh karena itu, diperlukan teknik pengolahan citra yang tepat agar model mampu mengenali karakteristik retakan secara akurat dalam berbagai kondisi lingkungan (Moreh et al., 2024).

Tantangan utama dalam teknologi *computer vision* untuk deteksi keretakan pada telur adalah bentuk alami telur yang tidak simetris dan permukaan yang melengkung kompleks. Kondisi ini menyulitkan deteksi visual, terutama untuk keretakan halus yang mungkin tidak terlihat dari satu sudut pandang saja. Tantangan ini semakin besar ketika telur berada dalam posisi miring atau tidak ideal saat bergerak diatas mesin konveyor. Dalam kondisi seperti ini, penggunaan kamera tunggal sering kali menghasilkan *blind spot* (titik buta), yaitu area permukaan telur yang tidak tertangkap oleh kamera, sehingga mengurangi akurasi sistem deteksi.

Sistem multi-kamera digunakan dalam analisis visual menangkap gambar dari berbagai sudut pandang. Pendekatan ini meningkatkan akurasi deteksi serta mengurangi risiko *blind spot*, terutama untuk objek berbentuk bulat seperti telur

ayam. Dalam sistem yang berbasis konveyor, multi-kamera memberikan informasi yang lebih menyeluruh mengenai kondisi permukaan telur, sehingga meningkatkan keandalan dalam proses klasifikasi keretakan.

Informasi dari beberapa kamera dapat secara optimal, diperlukan metode *blending* yang mampu menggabungkan gambar dari berbagai perspektif tanpa menyebabkan distorsi atau kehilangan detail penting. Salah satu pendekatan yang dapat diterapkan adalah *Side by side*, yang memungkinkan penggabungan gambar secara bertahap dalam berbagai frekuensi spasial. Dengan metode ini, gambar dari masing-masing kamera dapat diselaraskan secara optimal sehingga menghasilkan representasi yang lebih jelas dan akurat.

Tantangan selanjutnya algoritma yang populer dalam deteksi objek adalah *You Only Look Once* (YOLO), yang dikenal memiliki kemampuan deteksi dengan presisi tinggi. Algoritma ini membagi citra ke dalam beberapa *grid* dan secara langsung memprediksi *bounding box* serta kategori objek, sehingga proses deteksi menjadi lebih cepat dan efisien dibandingkan metode konvensional.

Algoritma YOLOv8 telah terbukti unggul dalam berbagai sistem deteksi, namun hasil pengujian pada deteksi telur masih menunjukkan adanya kesalahan signifikan pada *confusion matrix*, khususnya ketika model harus mengenali retakan tipis dengan pola tidak beraturan pada permukaan cangkang. Keterbatasan ini berdampak pada akurasi klasifikasi, di mana sebagian telur retak berpotensi terdeteksi sebagai telur utuh. Untuk mengatasi hal tersebut, penelitian ini mengusulkan peningkatan YOLOv8 melalui integrasi *Deformable Convolutional Networks version* 2 (DCNv2). Modul DCNv2 memungkinkan *kernel* konvolusi beradaptasi terhadap kontur permukaan telur, sehingga retakan halus dapat terdeteksi lebih akurat. Pendekatan ini diharapkan mampu meningkatkan akurasi deteksi sekaligus mengurangi kesalahan klasifikasi dibandingkan model YOLOv8 *baseline*.

Penelitian yang dilakukan oleh (Yang et al., 2023) dalam mendeteksi kerusakan pada telur menunjukkan bahwa penerapan YOLOv5 pada lebih dari 4000 citra telur mampu mencapai akurasi 96,3% dengan *presisi* 95,7% dan *recall* 94,8%. Hasil ini membuktikan bahwa metode berbasis *deep learning* lebih unggul dibanding metode candling tradisional, meskipun tantangan masih muncul dalam mendeteksi retakan halus pada kondisi pencahayaan dan posisi objek yang bervariasi (X. Li et al., 2025) mengembangkan YOLOv8 untuk mendeteksi pipa bor ditambang batubara dengan menambahkan modul *deformable convolution* pada *backbone*, yang berhasil meningkatkan *presisi* menjadi 94,2%, *recall* 93,1%, dan mAP50 95,6%, lebih tinggi dibandingkan model *baseline*.

Selain kedua penelitian tersebut, studi terbaru juga menunjukkan potensi besar pengembangan YOLOv8 diberbagai sektor industri. Sebagai contoh, penelitian oleh Qiu dan Ni (Qiu & Ni, 2025) mengusulkan peningkatan YOLOv8 untuk mendeteksi cacat pada permukaan aluminium, sementara studi lain (J. Li et al., 2024) memperkenalkan FSNB-YOLOv8 guna melakukan inspeksi cacat permukaan pada sistem industri berbasis *daring*. Penelitian lainnya (Liu & Cheng, 2024) berfokus pada optimalisasi YOLOv8 dalam deteksi cacat permukaan baja. Bahkan, pada bidang konstruksi, YOLOv8 yang telah dimodifikasi juga diterapkan untuk mendeteksi

kerusakan dan cacat permukaan bangunan. Seluruh penelitian tersebut secara konsisten menunjukkan bahwa penambahan modul seperti *deformable convolution*, *attention mechanism*, dan *multi scale feature extraction* mampu meningkatkan performa deteksi, terutama pada citra dengan kondisi visual yang kompleks. Berdasarkan hasil-hasil tersebut, penelitian ini mengembangkan model YOLOv8 dengan integrasi DCNv2 untuk memperkuat kemampuan ekstraksi fitur pada pola retakan halus pada telur ayam. Integrasi ini diharapkan dapat menghasilkan sistem deteksi dan klasifikasi yang lebih akurat, stabil, dan sesuai dengan kebutuhan industri peternakan unggas modern.

## 1.2 Landasan Teori

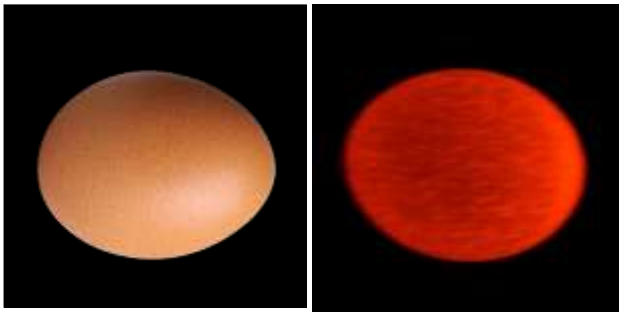
### 1.2.1 Telur Ayam

Produksi telur ayam memberikan kontribusi signifikan terhadap pemenuhan kebutuhan protein masyarakat dan mendukung berbagai industri pangan olahan. Manfaatnya yang beragam, mulai dari bahan konsumsi rumah tangga hingga bahan baku industri makanan dan farmasi, telah mendorong peningkatan permintaan telur secara berkelanjutan. Telur ayam merupakan salah satu komoditas hasil ternak yang memiliki nilai ekonomi tinggi dan berperan penting sebagai sumber protein hewani bagi masyarakat.

Indonesia termasuk salah satu negara penghasil telur ayam terbesar dikawasan Asia Tenggara, dengan daerah penyebaran meliputi Jawa, Sumatera, Sulawesi, Kalimantan, dan Bali, di mana sektor peternakan unggas berkembang pesat sebagai penunjang kebutuhan pangan nasional. Kondisi ini menunjukkan pentingnya pengelolaan produksi dan kualitas telur yang efektif serta memenuhi standar mutu yang tinggi. Proses penentuan kualitas telur pada umumnya masih dilakukan secara manual oleh tenaga *grader* yang bertugas menilai kelayakan telur berdasarkan kondisi fisik seperti cangkang, bentuk, dan kebersihan. Penilaian yang akurat terhadap kualitas telur, terutama dalam mendeteksi adanya keretakan pada cangkang, sangat penting untuk menjaga mutu, daya simpan, serta keamanan telur agar layak dikonsumsi masyarakat (De Oliveira-Boreli et al., 2023).

Citra telur ayam merupakan representasi visual dari kondisi luar telur yang digunakan sebagai objek awal dalam proses pengamatan dan analisis. Citra ini umumnya menampilkan bentuk, ukuran, dan permukaan cangkang telur secara langsung tanpa bantuan pencahayaan tambahan. Sementara itu, citra telur ayam *candling* diperoleh melalui teknik penerangan menggunakan sumber cahaya yang diarahkan ke dalam telur. Metode *candling* memungkinkan visualisasi kondisi bagian dalam telur, seperti keberadaan embrio, rongga udara, serta potensi cacat internal yang tidak dapat diamati secara kasat mata. Perbedaan karakteristik visual antara kedua citra ini memberikan informasi yang penting dalam proses inspeksi mutu telur ayam. bertujuan untuk memberikan gambaran perbandingan antara pengamatan telur secara konvensional dan pengamatan menggunakan teknik *candling*, sehingga dapat mendukung pemahaman terhadap tahapan pengolahan citra serta analisis

kualitas telur ayam yang dilakukan pada penelitian ini. Berikut ini Gambar 1 citra telur ayam dan citra telur ayam dengan metode *candling*.



**Gambar 1. Citra telur ayam dan Citra telur ayam *candling***

### 1.2.2 Visi komputer

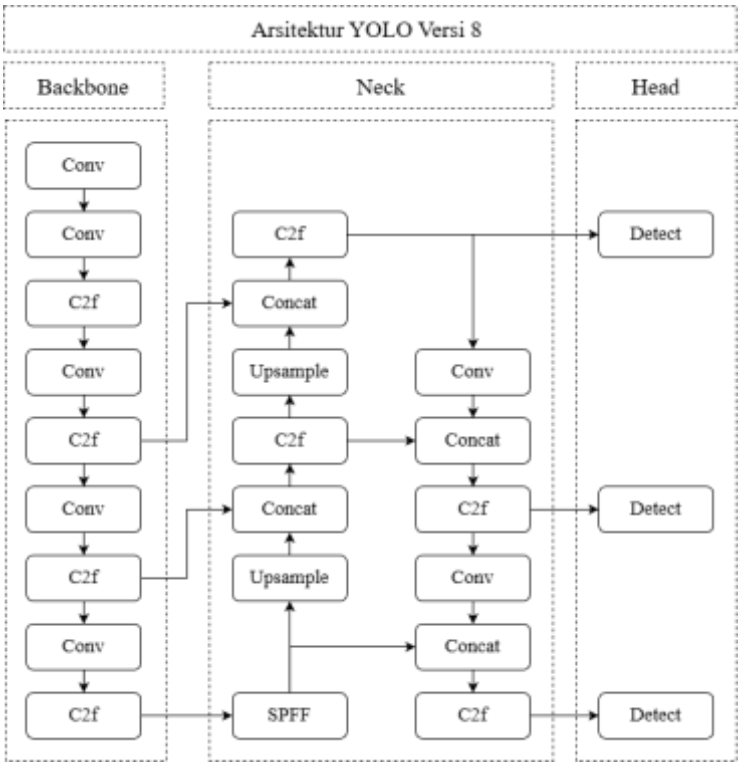
Visi komputer merupakan cabang dari kecerdasan buatan (*Artificial Intelligence*/AI) yang berfokus pada kemampuan mesin untuk memahami, menganalisis, dan menafsirkan informasi visual dari dunia nyata dalam bentuk gambar maupun video. Tujuan utama visi komputer adalah meniru kemampuan persepsi visual manusia sehingga komputer dapat mengenali objek, mendeteksi pola, serta mengambil keputusan berdasarkan data visual yang diperoleh. Dalam implementasinya, visi komputer mencakup berbagai proses seperti akuisisi citra, prapemrosesan, ekstraksi fitur, segmentasi, klasifikasi, dan deteksi objek. Melalui tahapan-tahapan tersebut, sistem dapat mengubah data visual mentah menjadi informasi yang bermakna dan dapat digunakan untuk berbagai aplikasi (Mahadevkar et al., 2022).

Teknologi visi komputer banyak diterapkan dalam berbagai bidang, seperti pengenalan wajah (*face recognition*), deteksi objek (*object detection*), pengenalan citra medis (*medical imaging*), pemantauan keamanan (*surveillance*), otomasi industri, dan pertanian cerdas (*smart agriculture*). Dalam konteks penelitian modern, visi komputer berperan penting dalam mendukung sistem berbasis pembelajaran mendalam (*deep learning*) untuk meningkatkan akurasi serta keandalan analisis visual.

Perkembangan pesat teknologi ini menjadikan visi komputer sebagai komponen utama dalam transformasi digital dan otomatisasi, karena kemampuannya dalam memberikan solusi cerdas berbasis pengenalan pola visual yang *efisien* dan *adaptif* terhadap berbagai kondisi lingkungan.

### 1.2.2 *You Only Look Once version* 8 (YOLOv8)

YOLOv8 merupakan algoritma deteksi objek yang dikembangkan oleh Ultralytics, yang dirancang untuk meningkatkan akurasi dan efisiensi dalam berbagai tugas deteksi. Arsitektur utama YOLOv8 terdiri atas tiga komponen utama, yaitu *backbone*, *neck*, dan *head*. Berikut ini Gambar 2 arsitektur YOLOv8.




**Gambar 2. Arsitektur YOLOv8**

Komponen *backbone* berfungsi untuk mengekstraksi fitur dari citra masukan melalui serangkaian lapisan konvolusi dan blok C2f (*Cross Stage Partial with* 2 *Convolutions*), yang bertujuan meningkatkan efisiensi komputasi tanpa mengurangi kemampuan jaringan dalam merepresentasikan fitur penting. Pada bagian akhir *backbone*, digunakan modul SPPF (*Spatial Pyramid Pooling Fast*) untuk memperluas *receptive field* dan menangkap konteks spasial yang lebih luas dari citra (Wang et al., 2025). Selanjutnya, bagian *neck* mengimplementasikan kombinasi *Feature Pyramid Network* (FPN) dan *Path Aggregation Network* (PAN) yang berperan penting dalam menggabungkan fitur dari berbagai tingkat resolusi. Dengan struktur ini, YOLOv8 mampu mempertahankan informasi detail dari level rendah sekaligus mengintegrasikannya dengan informasi semantik dari level tinggi, sehingga deteksi terhadap objek dengan ukuran dan bentuk yang bervariasi dapat dilakukan secara lebih efektif.

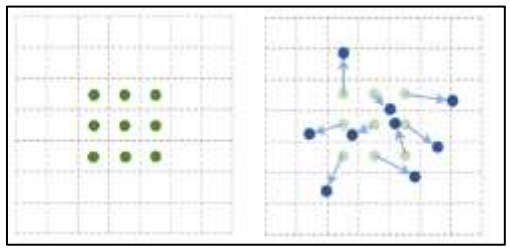
Bagian *head* terdiri atas lapisan deteksi multi skala yang menghasilkan prediksi berupa *bounding box*, *confidence score*, serta klasifikasi objek pada berbagai ukuran. Melalui desain ini, YOLOv8 dikenal memiliki kecepatan inferensi yang tinggi dan akurasi yang andal, menjadikannya salah satu model *state of the art* dalam bidang deteksi objek waktu nyata. Selain itu, *fleksibilitas* arsitektur YOLOv8 memungkinkan pengguna untuk menyesuaikan ukuran model sesuai kebutuhan, seperti varian YOLOv8n (*nano*), YOLOv8s (*small*), hingga YOLOv8x (*extrl arge*), sehingga dapat dioptimalkan untuk berbagai perangkat dan lingkungan komputasi.

### 1.2.3 *Deformable Convolutional Network Version* 2 (DCNv2)

*Deformable Convolutional Networks* v2 (DCNv2) merupakan pengembangan dari DCN yang dirancang untuk mengatasi keterbatasan konvolusi standar dalam menangkap deformasi bentuk dan variasi spasial pada objek. Pada konvolusi konvensional, posisi *sampling kernel* bersifat tetap mengikuti *grid*, seperti 3×3 atau 5×5, sehingga kurang *adaptif* terhadap objek yang miring, tidak beraturan, terdistorsi, atau hanya terlihat sebagian. DCNv2 memperbaiki keterbatasan ini dengan menambahkan dua komponen utama, yaitu *offset* dan *modulation*. *Offset* memungkinkan setiap titik *sampling* bergeser secara dinamis mengikuti kontur objek, sehingga kernel dapat mengambil informasi dari posisi yang lebih relevan (Yehia et al., 2025). Sementara itu, *modulation* memberikan bobot pentingnya tiap titik *sampling* melalui faktor sehingga model dapat memperkuat area yang informatif dan melemahkan area yang kurang penting. Secara matematis, *deformable convolution* v2 dinyatakan sebagai:

$$y(p_0) = \sum_{n=1}^{N} w_n \cdot x(p_0 + p_n + \Delta p_n) \cdot \Delta m_n \quad (1)$$

Dengan $p_0$sebagai pusat *kernel*, $p_n$ posisi *grid*, $\Delta p_n$*offset* yang dipelajari, dan $\Delta m_n$sebagai faktor modulasi. Melalui mekanisme ini, DCNv2 memberikan *fleksibilitas* geometris pada proses konvolusi sehingga model mampu menyesuaikan posisi *sampling* dan tingkat kontribusi fitur secara *adaptif*. Ilustrasi tersebut ditunjukkan pada Gambar 3 berikut:



**Gambar 3.** ***Ilustrasi Deformable Convolutional Network*** **(DCNv2)**

Keunggulan DCNv2 terletak pada kemampuannya menangkap detail lokal dan struktur objek yang kompleks tanpa menambah parameter jaringan secara signifikan, sehingga tetap *efisien* untuk aplikasi waktu nyata. Dibandingkan DCN generasi pertama, DCNv2 menghasilkan representasi fitur yang lebih stabil dan akurat karena adanya modulasi yang mengatur pentingnya setiap titik *sampling*. Ketika diterapkan pada model deteksi seperti YOLOv8, DCNv2 mampu meningkatkan kualitas ekstraksi fitur terutama pada kondisi pencahayaan rendah, bentuk objek yang tidak simetris, *motion blur*, atau citra yang sebagian tertutup. Dengan fleksibilitas *sampling* yang adaptif tersebut, model dapat lebih fokus pada bagian paling informatif dari citra.

### 1.2.4 Integrasi YOLOv8 Dengan DCNv2

Integrasi *Deformable Convolutional Network version* 2 (DCNv2) ke dalam arsitektur *You Only Look Once* (YOLO) bertujuan untuk meningkatkan kemampuan model dalam mengekstraksi fitur adaptif terhadap deformasi bentuk, variasi pencahayaan, dan kondisi visual yang kompleks seperti pada citra malam hari. Secara umum, YOLO terdiri atas tiga komponen utama, *backbone, neck,* dan *head*. *Backbone* berperan dalam ekstraksi fitur dasar, *neck* menggabungkan fitur dari berbagai skala menggunakan mekanisme seperti PAN atau FPN, dan *head* bertanggung jawab menghasilkan prediksi kelas, *bounding box*, serta nilai kepercayaan deteksi. Dengan menambahkan *Deformable Convolution version* 2 (DCNv2) pada salah satu atau beberapa bagian dari struktur ini, kemampuan jaringan untuk memahami konteks spasial dan bentuk objek dapat ditingkatkan secara signifikan.

Secara teknis, *deformable convolution* biasanya diintegrasikan pada bagian *backbone* karena lapisan ini bertugas mengekstraksi fitur lokal dan global dari citra *input*. Pada lapisan ini, DCNv2 menggantikan sebagian operasi konvolusi standar dengan versi *deformable* yang memiliki kemampuan adaptif terhadap posisi *sampling*. Beberapa penelitian juga menunjukkan bahwa penerapan DCNv2 pada *neck* dapat memperkuat kemampuan fusi fitur antar-skala, sehingga meningkatkan akurasi deteksi pada objek kecil seperti rambu lalu lintas yang berukuran relatif kecil didalam citra. Sementara itu, penerapan pada *head* bertujuan untuk meningkatkan presisi pada tahap akhir prediksi *bounding box* dan klasifikasi.

### 1.2.5 Image Processing

Pemrosesan gambar memainkan peran penting dalam menyempurnakan gambar sebelum menerapkan algoritma pengenalan pola, yang bertujuan untuk meningkatkan kualitas, mengurangi *noise*, mengoreksi pencahayaan, dan mengekstraksi fitur yang relevan. Teknik umum meliputi penyaringan, pemerataan *histogram*, deteksi tepi, dan operasi *morfologi*. Fondasi matematika yang kuat sangat penting untuk analisis gambar yang efektif, yang menjadi dasar untuk langkah-langkah berikutnya. Fondasi ini menentukan ruang warna dan model, yang merepresentasikan warna secara matematis (Manakitsa et al., 2024).

Pengolahan Citra adalah ilmu yang berfokus pada pengolahan dan manipulasi gambar digital. Tujuan utama dari pengolahan citra adalah untuk meningkatkan kualitas gambar, mengubah atribut gambar, serta mengekstraksi informasi yang berguna dari citra tersebut. Proses pengolahan citra dimulai dengan akuisisi gambar dalam format digital menggunakan perangkat seperti kamera digital. Gambar digital terdiri dari sekumpulan piksel atau elemen gambar yang membentuk tampilan visual yang dapat dilihat oleh mata manusia. Setiap piksel memiliki nilai *intensitas* yang merepresentasikan warna atau tingkat kecerahan pada posisi tertentu dalam citra. Model warna seperti RGB, HSI, dan HSV mendefinisikan warna secara tepat menggunakan variabel, yang membentuk ruang warna. Model RGB terdiri dari komponen merah, hijau, dan biru. Model ini menggabungkan tingkat intensitas komponen-komponen ini untuk menciptakan warna. Kekuatan penuh dari ketiganya

menghasilkan warna putih, sedangkan ketidakhadirannya menghasilkan warna hitam (Manakitsa et al., 2024).

Salah satu operasi dasar adalah peningkatan kualitas gambar, yang mencakup pembersihan gambar dari *noise* atau gangguan, meningkatkan kejelasan dan kontras, serta ketajaman gambar. Transformasi gambar juga penting, di mana teknik seperti rotasi, skala, dan pemangkasan atau *cropping* digunakan untuk mengubah ukuran dan orientasi gambar. Selain itu, proses segmentasi membagi gambar menjadi beberapa bagian yang lebih kecil atau mengidentifikasi objek dan wilayah tertentu dalam gambar. Melalui berbagai operasi ini, pengolahan citra dapat memberikan hasil yang lebih informatif dan relevan sesuai dengan kebutuhan analisis yang diinginkan.

### 1.2.6 Deteksi Objek

Deteksi objek merupakan salah satu tugas penting yang bertujuan untuk mengidentifikasi objek-objek dalam gambar atau video. Salah satu tantangan utama dalam deteksi objek adalah variasi ukuran objek yang ada dalam gambar (Akita & Ukita, 2023). Deteksi objek, deteksi wajah, dan pengenalan warna adalah beberapa contoh, yang menggambarkan ketergantungan sistem pada pola tertentu untuk ekstraksi informasi, salah satu bentuk kecerdasan manusia adalah kemampuannya dalam mengenali objek disekitarnya, seperti mengenali sesama manusia. Manusia menggunakan mata sebagai indra penglihatan untuk menangkap citra objek, yang kemudian diproses dan disimpan dalam memori otak. Perkembangan teknologi saat ini memungkinkan mesin untuk belajar mengenali objek seperti manusia. Mesin membutuhkan kecerdasan buatan untuk mengenali dan mengklasifikasikan *citra* digital suatu objek (Manakitsa et al., 2024).

### 1.2.7 Konveyor

Konveyor merupakan sistem transportasi otomatis yang berperan penting dalam mendukung proses inspeksi mutu dan pemilahan berbasis *computer vision*. Konveyor memastikan aliran objek yang stabil, teratur, dan berkelanjutan sehingga setiap objek dapat melewati titik pengamatan kamera dengan kecepatan, jarak yang terkontrol. Penggunaan konveyor menggantikan pemindahan manual yang cenderung lambat dan tidak konsisten, sekaligus memungkinkan penanganan volume objek yang tinggi dengan risiko kerusakan dan kontaminasi yang minimal. Selain itu, pengaturan kecepatan dan kepadatan objek pada konveyor sangat memengaruhi kualitas citra dan akurasi sortasi otomatis, menjadikan konveyor sebagai komponen kunci dalam meningkatkan efisiensi, kapasitas, dan konsistensi sistem inspeksi visual (Siregar et al., 2023).

Sistem konveyor yang digunakan untuk proses pemeriksaan telur menggunakan metode *candling*. Pada bagian atas terlihat *belt* konveyor yang bergerak secara kontinu untuk membawa telur melewati area inspeksi. Konveyor digerakkan oleh dua roda penggerak (*pulley*) disisi kiri dan kanan yang tersambung dengan motor, sehingga belt dapat berjalan secara stabil. Telur-telur ditempatkan

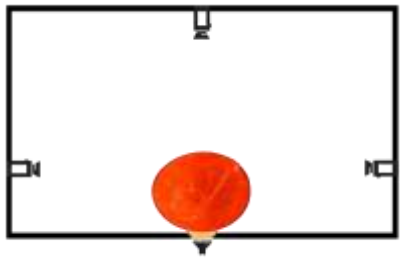
berurutan diatas *belt* dan bergerak mengikuti arah putaran konveyor. Dibagian tengah konveyor terdapat sebuah ruang pemeriksaan yang diberi penutup berwarna hitam. Di dalam ruang tersebut terdapat sumber cahaya dari bawah, yang berfungsi untuk melakukan *candling*, yaitu menyorot telur dari bawah agar struktur dalam telur terlihat jelas. Tulisan dan gambarnya menunjukkan bahwa ketika cahaya menembus telur, kamera yang berada di bagian atas akan menangkap citra telur. Kamera samping kiri dan kanan juga digunakan untuk mengambil gambar dari berbagai sudut sehingga proses deteksi menjadi lebih akurat. Ketika telur berada tepat diatas cahaya, tampak bahwa isinya terlihat lebih jelas misalnya retakan, bercak darah, atau kelainan lainnya dapat divisualisasikan. Proses ini memungkinkan sistem untuk melakukan klasifikasi berdasarkan kondisi telur, seperti telur baik atau telur retak. Konveyor berfungsi memastikan satu per satu melewati area iluminasi, sehingga kamera dapat mengambil gambar secara konsisten tanpa gangguan. Bisa dilihat pada Gambar 4 berikut ini :

**Gambar 4. Skenario Pengambilan Data**

Secara keseluruhan, Gambar 4 tersebut menunjukkan integrasi antara *belt* konveyor, sumber cahaya *candling*, dan sistem multi-kamera untuk mendukung proses inspeksi telur secara otomatis dan efisien. Sistem seperti ini umum digunakan dalam industri unggas untuk meningkatkan kecepatan, akurasi, dan *efisiensi* pemilah telur.

### 1.2.8 Multi-Kamera

Sistem multi-kamera adalah konfigurasi yang menggunakan lebih dari satu kamera untuk menangkap citra objek dari beberapa sudut secara bersamaan. Pada proses inspeksi telur berbasis konveyor dan *candling*, penerapan multi-kamera memberikan keuntungan signifikan karena setiap kamera dapat merekam bagian telur yang berbeda sehingga informasi visual yang diperoleh jauh lebih lengkap dibandingkan penggunaan satu kamera saja (Nazeri et al., 2025). Gambar 5 Ilustrasi multi-kamera tersebut ditunjukkan pada Gambar berikut:

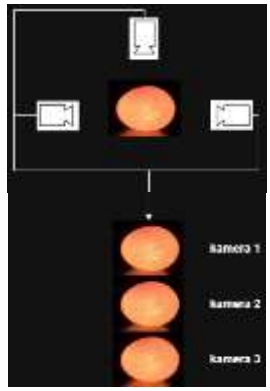


**Gambar 5. Ilustrasi pengambilan data multi-kamera**

Sistem ini, satu kamera ditempatkan di bagian atas untuk menangkap *citra* utama dari telur yang sedang disinari dari bawah. Kamera atas sangat penting untuk menangkap pola retakan, kontur cangkang, dan struktur bagian dalam telur yang tampak melalui cahaya *candling*. Sementara itu, dua kamera lain diposisikan disisi kiri dan kanan. Kamera samping berfungsi untuk menangkap detail tambahan pada bagian samping telur yang tidak terlihat dari kamera atas, seperti retakan melintang atau kerusakan struktural diarea yang biasanya tertutup bayangan. Dengan adanya multi-kamera, sistem mendapatkan perspektif 3 dimensi secara virtual, karena setiap sisi telur terdokumentasi secara simultan. Data dari tiga kamera kemudian dapat digabungkan atau dianalisis secara terpisah untuk meningkatkan akurasi algoritma pendeteksian, misalnya YOLOv8 atau model modifikasi lainnya. Pendekatan ini sangat efektif terutama dalam mendeteksi retakan tipis, pola yang tidak beraturan, atau area redup yang sulit diidentifikasi hanya dari satu sudut. Melalui penggunaan multi-kamera, proses inspeksi menjadi lebih komprehensif, stabil, dan andal, sehingga meminimalkan kesalahan klasifikasi serta meningkatkan keakuratan sistem dalam membedakan telur baik dan telur retak pada kecepatan konveyor yang berbeda.

### 1.2.9 *Stitching* video

Sistem berbasis konveyor dalam penggunaan multi-kamera bertujuan untuk memperoleh informasi visual yang lebih lengkap dari objek yang bergerak, seperti telur yang diperiksa dengan metode *candling*. Setiap kamera menangkap gambar dari sudut yang berbeda misalnya kamera atas, kamera sisi kiri, dan kamera sisi kanan. Agar data dari berbagai sudut ini dapat dianalisis secara efisien oleh model deteksi seperti YOLOv8 atau algoritma lainnya, rekaman dari ketiga kamera dapat digabungkan menggunakan metode video *stitching side by side*. Ilustrasi *stitching* tersebut ditunjukkan pada Gambar 6 berikut:




**Gambar 6. Ilustrasi *stitching side by side***

Teknik *stitching side by side* adalah proses menggabungkan beberapa video ke dalam satu *frame* yang lebih lebar, dengan cara menempatkan setiap video disamping satu sama lain secara *horizontal*. Artinya, video dari kamera atas, kiri, dan kanan disusun berdampingan dalam satu tampilan yang utuh. Proses penggabungan ini tidak memerlukan pemetaan perspektif kompleks seperti pada *stitching* panorama yang diperlukan hanyalah menyesuaikan ukuran *frame* dan menyinkronkan waktu pengambilan gambar agar ketiga video tersusun rapi dalam satu *layout*. Dengan menggunakan *stitching side by side*, sistem memperoleh satu video gabungan yang berisi tiga sudut pandang berbeda secara simultan (Lyu et al., 2019). *Output* gabungan ini lebih mudah diproses oleh model deteksi karena semua informasi visual tersedia dalam satu *frame* sehingga mempermudah pelabelan *dataset*, mempercepat proses *anotasi*, dan mengurangi risiko ketidaksinkronan antar *frame*. Selain itu, *stitching* juga mempermudah proses *debugging* dan *monitoring*, karena operator cukup melihat satu monitor untuk mengevaluasi kondisi objek dari berbagai sudut.

Penggabungan video ini biasanya dilakukan melalui perangkat lunak atau program seperti *Open* CV, di mana *frame* dari masing-masing kamera diambil secara *real time* lalu diletakkan berdampingan sesuai urutan yang ditentukan kiri, atas dan kanan. Setiap *frame* disejajarkan dalam ukuran dan orientasi seragam, sehingga hasilnya tampak seperti satu bidang video besar. Metode ini menghasilkan tampilan multi sudut yang komprehensif dan dapat meningkatkan akurasi sistem deteksi keretakan telur, karena model memperoleh lebih banyak informasi visual dalam satu kali pemrosesan. Dengan demikian, teknik *stitching side by side* membuat sistem multi-kamera menjadi lebih praktis dan efisien, sekaligus memastikan seluruh permukaan telur teramati jelas dalam satu *frame* gabungan.

### 1.2.10 *Confusion Matrix*

*Confusion matrix* memberikan informasi rinci mengenai kinerja model dengan menunjukkan jumlah prediksi yang benar dan salah untuk setiap kategori (Sathyanarayanan, 2024). Hal ini menunjukkan evaluasi yang lebih jelas terhadap

performa model dalam klasifikasi, Dalam evaluasi sistem deteksi objek, ketika suatu objek yang tercantum pada *ground truth* tidak berhasil dikenali atau diprediksi oleh model, objek tersebut dianggap sebagai bagian dari *background*. Kondisi ini diklasifikasikan sebagai *false negative*, karena keberadaan objek sebenarnya ada, tetapi tidak direspons oleh model. Secara konseptual, mekanisme ini sejalan dengan pendekatan *No Predicted Label* (NPL), yaitu kondisi ketika sebuah *instance ground truth* tidak memiliki pasangan prediksi. Sebaliknya, *No True Label* (NTL) digunakan untuk merepresentasikan prediksi model yang tidak memiliki padanan *ground truth*. Kedua konsep ini tidak diperlakukan sebagai kelas objek yang sesungguhnya, melainkan sebagai alat evaluasi yang fungsinya setara dengan *background* dalam *confusion matrix.* Dengan demikian, *background* berperan sebagai kategori implisit yang memungkinkan pencatatan kesalahan dan keberhasilan model secara sistematis, sehingga kinerja deteksi objek khususnya dalam mengidentifikasi keterbatasan model dalam mendeteksi objek tertentu dapat dianalisis secara lebih terstruktur dan komprehensif. *Confusion matrix* merupakan metrik *machine learning* yang menunjukkan jumlah:

1. *True Positive* (TP): label "*True*", prediksi "*True*".
2. *False Positive* (FP): label "*False*", prediksi "*True*".
3. *True Negative* (TN): label "*True*", prediksi "*False*".
4. *False Negative* (FN): label "*False*", prediksi "*False*".

**Tabel 1. *Confusion Matrix***

| | | *Actual Values* | |
|---|---|---|---|
| | | *Positive* | *Negative* |
| *Predicted Values* | *Positive* | TP | FP |
| | *Negative* | FN | TN |

*Metrik* evaluasi yang digunakan adalah *precision*, *recall*, dan mAP.

*1. Precision* menggambarkan tingkat keakuratan model dalam memberikan prediksi yang sesuai dengan data yang diharapkan. *Precision* dihitung sebagai rasio antara jumlah prediksi positif yang benar dengan total data yang di prediksi sebagai positif. Dengan kata lain, *precision* menunjukkan seberapa banyak dari prediksi positif yang benar benar merupakan data positif. Nilai *precision* dapat diperoleh dengan persamaan (2).

$$Precision = \frac{TP}{TP + FP} \tag{2}$$

2. *Recall* menunjukkan kemampuan model dalam mengidentifikasi Kembali informasi yang relevan. *Recall* dihitung sebagai rasio antara jumlah prediksi positif yang benar terhadap total data yang benar-benar positif. Dengan kata lain, *recall* mengukur sejauh mana model berhasil mendeteksi data positif secara akurat. Nilai *recall* dapat diperoleh dengan persamaan (3).

$$Recall = \frac{TP}{TP + FN} \tag{3}$$

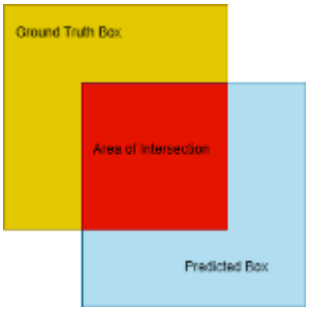
3. mAP adalah metrik evaluasi yang digunakan untuk mengukur performa model deteksi objek. Metrik ini menghitung rata-rata presisi dari semua kelas yang terdeteksi. Presisi sendiri menggambarkan seberapa akurat prediksi model terhadap objek yang benar-benar ada. Nilai mAP dapat diperoleh dengan persamaan (4)

$$mAP = \frac{1}{N}\sum_{i=1}^{n} AP_i \tag{4}$$

*4. Intersection over Union* (IoU) merupakan metrik yang digunakan untuk menilai tingkat kesesuaian antara *bounding box* hasil prediksi dengan *bounding box* sebenarnya. Metrik ini bekerja dengan membandingkan luas area yang menjadi tumpang tindih antara kedua kotak (*Area of Intersection*) dengan luas gabungan keduanya (*Area of Union*). Semakin besar area yang saling menutupi, semakin akurat prediksi posisi objek yang dilakukan model. IoU digunakan sebagai dasar untuk menentukan apakah suatu prediksi dianggap benar atau tidak berdasarkan ambang batas tertentu.

$$IoU = \frac{\text{Area of Intersection}}{\text{Area of Union}} \tag{5}$$

Memberikan gambaran yang mengenai cara perhitungan IoU, ilustrasi pada Gambar 7 menunjukkan posisi *ground truth box* dan *predicted box* beserta area tumpang tindih (*intersection*) diantara keduanya. Area yang ditandai dengan warna merah merupakan bagian yang menjadi *intersection*, sedangkan keseluruhan area yang dibentuk oleh kedua kotak tersebut merupakan *union*. Nilai IoU akan meningkat ketika area *intersection* semakin besar dibandingkan luas *union*, yang berarti prediksi model semakin mendekati anotasi sebenarnya.




**Gambar 7. *Intersection over Union* (IoU)**

## 1.3 Rumusan Masalah

Berdasarkan latar belakang masalah diatas maka rumusan masalah pada penelitian ini adalah sebagai berikut.

1. Bagaimana sistem ini dapat mengatasi tantangan dalam memperoleh citra telur secara menyeluruh dari berbagai sudut pandang, sehingga dapat meningkatkan deteksi dan klasifikasi keretakan pada seluruh bagian telur?
2. Bagaimana meningkatkan performa algoritma YOLOv8 dalam mendeteksi dan Klasifikasi keretakan pada telur ayam hasil tangkapan sistem multi-kamera?

## 1.4 Tujuan dan Manfaat

### 1.4.1 Tujuan

Adapun tujuan yang ingin dicapai pada penelitian ini yaitu sebagai berikut.

1. Mengembangkan sistem multi-kamera yang mampu menangkap citra telur secara menyeluruh dari berbagai sudut pandang untuk meningkatkan akurasi deteksi keretakan.
2. Mengoptimalkan algoritma YOLOv8 dalam mendeteksi keretakan telur ayam agar diperoleh hasil deteksi yang lebih akurat.

### 1.4.2 Manfaat

Manfaat yang dapat diperoleh dari penelitian yaitu sebagai berikut.

1. Mengembangkan sistem multi-kamera yang mampu menangkap citra telur secara menyeluruh dari berbagai sudut pandang untuk meningkatkan akurasi deteksi dan klasifikasi keretakan telur ayam.
2. Sebagai bahan rujukan dalam pengembangan ilmu pengetahuan dan teknologi bagi para peneliti bidang serupa dan akademik.

## 1.5 Batasan Masalah

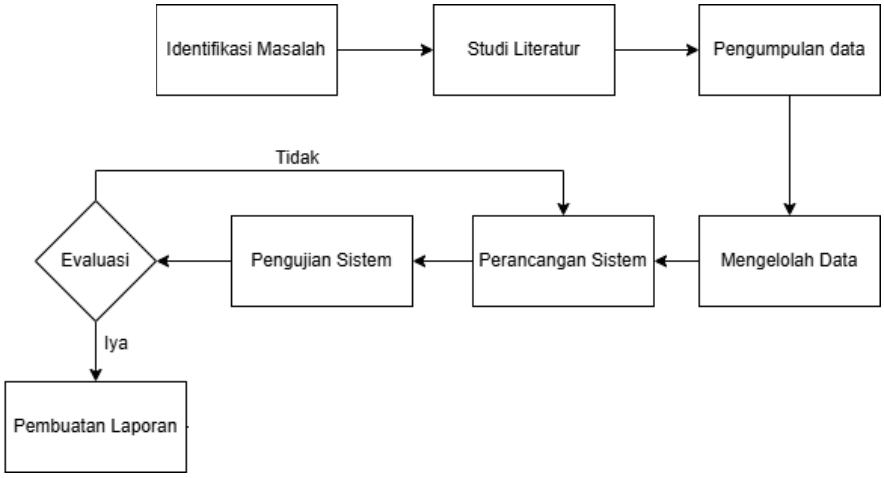
Adapun batasan masalah pada penelitian ini yaitu:

1. Penelitian di fokuskan pada deteksi kualitas telur ayam ras, khususnya dalam proses identifikasi kondisi fisik telur.
2. Proses deteksi hanya mencakup kondisi fisik eksternal, seperti adanya retakan pada permukaan cangkang telur.
3. Sistem pengambilan citra menggunakan tiga kamera yang di posisikan pada sisi kiri, kanan, dan atas untuk memperoleh citra telur secara menyeluruh dari berbagai sudut pandang.
4. Pengujian dilakukan pada tiga variasi kecepatan: yaitu 0,03 m/s, 0,04 m/s dan 0,05 m/s, untuk menganalisis pengaruh kecepatan terhadap performa deteksi dan klasifikasi.

# BAB II
# METODE PENELITIAN

## 2.1 Tahapan Penelitian

Penelitian ini merupakan penelitian dengan jenis eksperimental. Pengumpulan data berupa studi literatur dari jurnal dan artikel ilmiah sehingga ditemukan ruang lingkup masalah pada penelitian yang dilakukan, berikut disajikan tahapan penelitian:




**Gambar 8. Tahapan Penelitian**

### 2.1.1 Identifikasi Masalah

Identifikasi penelitian ini, terdapat tantangan dalam meningkatkan performa klasifikasi keretakan pada telur ayam ras, terutama akibat sudut pandang pengambilan citra. Selain itu, perbedaan sudut pandang kamera dapat menyebabkan distorsi bentuk dan pergeseran posisi retakan yang berpotensi menurunkan performa deteksi. Oleh karena itu, diperlukan sistem multi-kamera yang mampu menangkap citra telur secara menyeluruh dari berbagai sisi serta penerapan algoritma deteksi berbasis YOLOv8 yang dioptimalkan agar mampu mengenali pola retakan tipis dengan lebih akurat dan konsisten.

### 2.1.2 Studi Literatur

Tahapan studi literatur merupakan proses pengumpulan informasi, teori, serta referensi yang relevan dengan topik penelitian, yaitu deteksi keretakan pada telur ayam ras menggunakan sistem multi-kamera berbasis visi komputer. Pada tahap ini, peneliti menelaah berbagai sumber seperti jurnal ilmiah, buku, dan hasil penelitian terdahulu yang berkaitan dengan penerapan teknologi visi komputer, pengolahan citra, serta algoritma deteksi objek YOLOv8. Selain itu, pada tahap ini juga dilakukan penentuan metode yang tepat untuk menyelesaikan permasalahan dan tantangan yang dihadapi, khususnya dalam meningkatkan akurasi deteksi keretakan telur. Penelitian ini menggunakan sistem multi-kamera untuk memperoleh citra telur dari berbagai sudut pandang, serta menerapkan algoritma YOLOv8 dalam proses deteksi dan klasifikasi kondisi telur berdasarkan tingkat keretakan pada permukaan cangkangnya telur.

### 2.1.3 Pengumpulan Data

Tahap ini merupakan proses pengumpulan *dataset* yang akan digunakan dalam penelitian. *Dataset* yang dikumpulkan berupa citra telur ayam ras yang diambil menggunakan tiga kamera, yaitu kamera sisi kiri, kanan, dan atas, agar diperoleh tampilan telur secara menyeluruh dari berbagai sudut pandang. Pengambilan citra dilakukan pada kondisi pencahayaan terkendali dengan variasi waktu pencahayaan sebesar 0,03 m/s, 0,04 m/s dan 0,05 m/s untuk memperoleh kualitas gambar yang optimal. Setiap citra telur dikategorikan ke dalam dua kelas, yaitu telur utuh dan telur retak, berdasarkan kondisi fisik cangkangnya. *Dataset* yang terkumpul kemudian akan digunakan sebagai data pelatihan (*training set*) dan data pengujian (*testing set*) untuk melatih serta menguji performa model deteksi berbasis algoritma YOLOv8 dalam mengenali dan mengklasifikasikan tingkat keretakan telur ayam.

### 2.1.4 Mengelola Data

Tahap ini dilakukan pengolahan data citra telur ayam untuk memperoleh informasi visual yang relevan, seperti pola retakan, tekstur, dan intensitas cahaya pada permukaan cangkang telur. Proses ini mencakup beberapa langkah pada tahap pra pemrosesan (*preprocessing*), yaitu anotasi (*notasi*), pelabelan (*labelling*), *augmentation* data, pengubahan ukuran citra (*resize*), serta pembagian *dataset* (*split data*). Tahapan ini bertujuan untuk meningkatkan kualitas citra dan memastikan data yang digunakan memiliki konsistensi visual sehingga model YOLOv8 dapat dilatih secara optimal dalam mendeteksi serta mengklasifikasikan keretakan pada telur ayam.

### 2.1.5 Perancangan Sistem

Tahapan perancangan sistem merupakan tahap di mana dilakukan seluruh proses perancangan terhadap sistem yang akan dikembangkan untuk mendeteksi keretakan pada telur ayam. Pada tahap ini, sistem dirancang secara rinci dan terstruktur agar mampu mengintegrasikan tiga kamera (kiri, kanan dan atas) guna memperoleh citra telur dari berbagai sudut pandang. Perancangan dilakukan dengan tujuan untuk memastikan sistem dapat bekerja secara optimal dalam mengidentifikasi keretakan cangkang telur secara menyeluruh dan meningkatkan akurasi deteksi melalui implementasi algoritma YOLOv8 yang telah di optimalkan.

### 2.1.6 Pengujian dan Analisis

Tahap pengujian dilakukan dengan menggunakan *dataset* uji terpisah untuk mengevaluasi kinerja sistem yang telah dikembangkan. Setiap citra dari *dataset* uji dianalisis untuk mendeteksi dan mengklasifikasikan kondisi telur ayam, khususnya dalam mengidentifikasi adanya keretakan pada cangkang. Pengujian dilakukan dengan mempertimbangkan variasi kecepatan (0,03 m/s, 0,04 m/s dan 0,05 m/s) berguna menilai konsistensi sistem terhadap perubahan kecepatan.

### 2.1.7 Evaluasi

Evaluasi sistem dilakukan untuk menilai kinerja dan efektivitas sistem dalam mendeteksi serta mengklasifikasikan kualitas telur ayam ras secara otomatis, khususnya dalam mengidentifikasi kondisi fisik cangkang seperti telur retak dan telur

utuh. Proses evaluasi ini bertujuan untuk mengetahui sejauh mana sistem mampu memberikan prediksi yang akurat dan konsisten berdasarkan data uji yang telah disiapkan. Pengujian dilakukan dengan membandingkan hasil prediksi sistem terhadap data *ground truth*, kemudian dianalisis menggunakan metrik evaluasi standar dalam bidang deteksi objek, yaitu *precision* untuk mengukur ketepatan prediksi, *recall* untuk menilai kemampuan sistem dalam mendeteksi seluruh objek yang relevan, serta *mean Average Precision* (mAP) untuk memberikan gambaran menyeluruh mengenai performa sistem pada berbagai tingkat ambang kepercayaan. Melalui penggunaan metrik-metrik tersebut, keandalan dan akurasi sistem dapat divalidasi secara komprehensif guna memastikan bahwa sistem mampu memberikan hasil prediksi kualitas telur yang optimal dan layak diterapkan pada skenario penggunaan nyata.

### 2.1.8 Pembuatan Laporan

Bagian ini akan membahas hasil eksperimen dan analisis yang dilakukan, menginterpretasikan temuan temuan yang ada, serta membandingkan hasil yang diperoleh dengan penelitian sebelumnya.

## 2.2 Sumber Data

Proses pengumpulan data pada penelitian ini dilakukan dengan mengambil citra telur ayam ras dalam dua kondisi, yaitu telur utuh dan telur retak. Data diperoleh dari rekaman video telur yang bergerak diatas mesin konveyor, menggunakan kamera *Logitech* resolusi 1080 piksel dengan kecepatan perekaman 30-60 *frame* per detik (*fps*). Kamera ditempatkan pada ketinggian 15 cm dari objek untuk mendapatkan sudut pandang yang optimal terhadap permukaan telur.

Seluruh proses pengambilan citra dilakukan secara langsung di Laboratorium *Artificial Intelligence and Machine Perception* (AIMP). Sistem akuisisi data menggunakan tiga kamera digital resolusi *Full HD* (1080 × 720 piksel) yang di posisikan pada sisi kiri, kanan, dan atas telur untuk memperoleh tampilan citra yang menyeluruh dari berbagai sudut pandang. Setiap kamera terhubung ke sistem komputer yang dikendalikan secara sinkron (*synchronous control*) agar setiap pengambilan citra dilakukan secara bersamaan pada kondisi yang sama.

## 2.3 Perangkat Penelitian

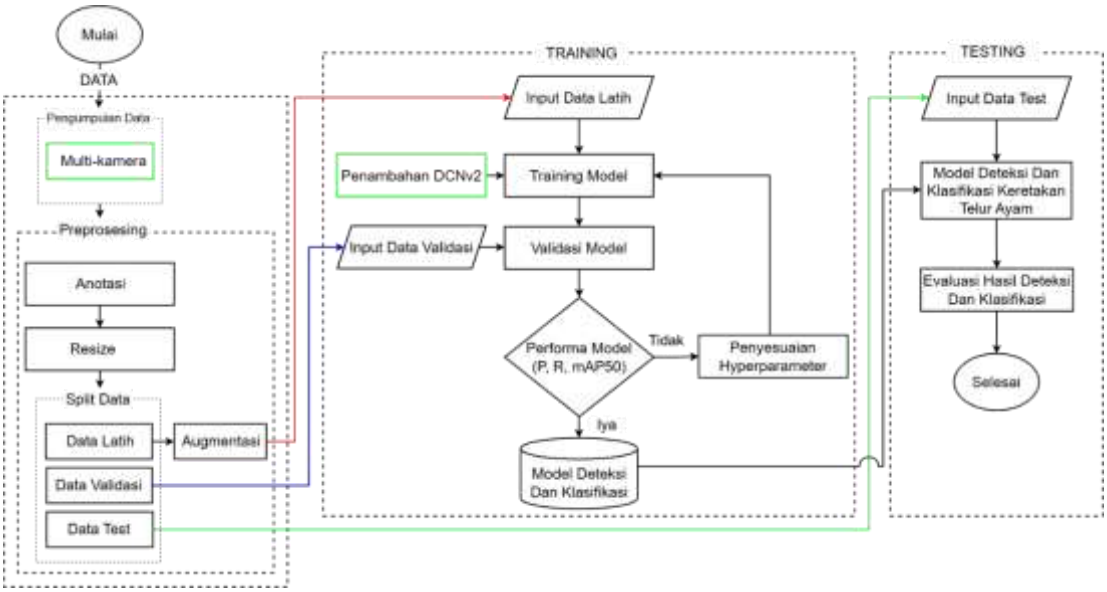
Adapun perangkat perangkat yang digunakan dalam menunjang penelitian ini yaitu sebagai berikut:

1. Perangkat Keras (*Hardware*)
    a. Personal *Computer* (PC)
    b. konveyor
    c. dinamo rs 550
    d. lampu *led*
    e. Logitech 920 *HD* Webcam, 1080 piksel
    f. draw.io
    g. Microsof Office
2. Perangkat Lunak (*Software*)
    a. Sistem Operasi *Windows* 10-64 bit

b. Bahasa Pemrograman Python
c. Jupyter *Notebook*
d. Kagell GPU P100
e. Roboflow
f. Microsoft *Office*

## 2.4 Rancangan Sistem

Alur perancangan sistem pada penelitian ini ditunjukkan pada Gambar 9 berikut:




**Gambar 9. *Flowchart* Rancangan Sistem**

### 2.4.1 Data

**1. Pengumpulan Data**

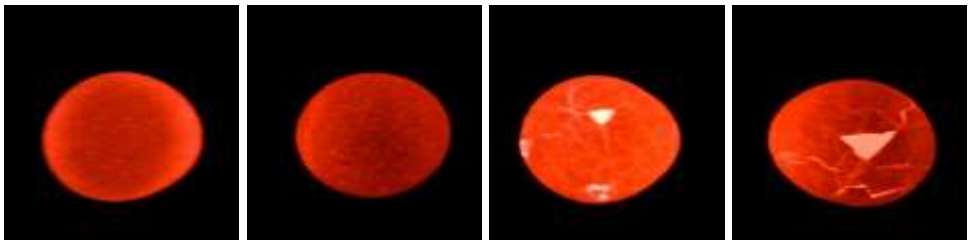
Pengumpulan data pada penelitian ini dilakukan dengan merekam video telur ayam ras yang bergerak diatas mesin konveyor menggunakan kamera Logitech resolusi 1080p dengan kecepatan 30 *frame* per detik (fps). Proses perekaman dilakukan secara langsung di Laboratorium *Artificial Intelligence and Machine Perception* (AIMP) untuk memperoleh data citra telur dalam dua kondisi, yaitu telur utuh dan telur retak.

Video direkam dengan tiga variasi kecepatan konveyor, yaitu 0,03 m/s, 0,04 m/s, dan 0,05 m/s, guna mensimulasikan kondisi operasional industri pemrosesan telur yang sesungguhnya. Kamera ditempatkan pada ketinggian 15 cm dari permukaan konveyor untuk mendapatkan sudut pandang yang optimal terhadap bentuk dan tekstur cangkang telur. Gambar kamera tersebut ditunjukkan pada Gambar 10 berikut ini :

**Gambar 10. Logitech 920 *HD* Webcam, 1080 Piksel**

Selain itu, sistem pengambilan data menggunakan tiga kamera digital beresolusi *Full HD* (1080 × 720 piksel) yang di posisikan pada sisi kiri, kanan, dan atas telur untuk memperoleh tampilan citra secara menyeluruh dari berbagai sudut pandang. Setiap kamera dihubungkan ke komputer utama dan dikendalikan secara sinkron, sehingga setiap *frame* yang diambil berada pada kondisi waktu perekaman yang sama

Video yang diperoleh memiliki durasi 5 hingga 10 detik untuk setiap percobaan, kemudian diekstrak menjadi gambar (*frame*) dengan interval 1 *frame* setiap 1 detik. Hasil ekstraksi citra disimpan dalam format .jpg dan digunakan sebagai *dataset* penelitian. Proses pengumpulan data ini menghasilkan sebanyak 1.691 citra telur ayam ras yang terdiri atas dua kelas, yaitu telur utuh dan telur retak.



**Gambar 11. Hasil Ekstrak Video Menjadi Gambar**

**2. *Preprocessing***

Setelah proses pengumpulan data dilakukan, dilanjutkan ke tahap pra pemrosesan (*preprocessing*), untuk mempersiapkan gambar sebelum diproses lebih lanjut dengan tujuan untuk memperoleh hasil yang optimal. Tahapan dalam proses ini meliput:

a. Anotasi

Pada Tahap anotasi data merupakan proses pemberian label pada citra telur ayam ras yang akan digunakan sebagai data latih model deteksi objek. Proses anotasi dilakukan menggunakan aplikasi berbasis web Roboflow. Objek yang menjadi fokus dalam penelitian ini adalah telur ayam ras dengan dua kelas utama, yaitu Telur Utuh dan Telur Retak. Pada setiap citra, objek telur diberi *bounding box* yang menandai lokasi serta batas area objek yang akan dideteksi oleh model. Proses pelabelan dilakukan secara manual untuk memastikan ketepatan penentuan posisi dan ukuran bounding box sesuai dengan kondisi fisik telur pada citra. Setiap objek kemudian diklasifikasikan ke dalam kelas yang telah ditentukan berdasarkan kondisi cangkangnya, yaitu telur utuh atau telur retak. Tujuan utama dari tahap anotasi ini adalah menghasilkan data berlabel yang akurat, sehingga model deteksi objek dapat mempelajari karakteristik visual masing-masing kelas dengan baik. Setelah seluruh citra selesai dianotasi, data diekspor dalam format YOLOv8 (.txt) yang kompatibel

secara langsung dengan *framework* Ultralytics YOLOv8 berbasis PyTorch, sehingga siap digunakan pada tahap pelatihan dan evaluasi model.


**Gambar 12. Anotasi Data**

b. *Resize*

Proses *resize* atau perubahan ukuran gambar dilakukan dengan mengubah citra telur ayam ras dari resolusi asli 1080 × 720 piksel menjadi 640 × 640 piksel. Perubahan ukuran ini bertujuan untuk memperkecil ukuran data agar proses pelatihan model dapat berjalan lebih cepat dan efisien dalam penggunaan memori. Dengan melakukan *resize*, model tetap mampu mengenali fitur fitur penting pada citra telur, seperti bentuk dan pola retakan pada cangkang, tanpa kehilangan informasi visual yang diperlukan untuk proses deteksi dan klasifikasi.

c. *Split* Data

Setelah dilakukan proses *resize*, tahap selanjutnya adalah pembagian *dataset* (data *splitting*) yang dilakukan untuk membagi kumpulan citra telur ayam ras ke dalam tiga bagian, yaitu data *training*, data validasi, dan data *testing*. Pembagian data ini bertujuan untuk melatih, memvalidasi, serta menguji kinerja model dalam proses deteksi dan klasifikasi kondisi telur ayam ras menggunakan metode yang diusulkan.

1. Data *Training*

Data *training* digunakan untuk melatih model dalam mengenali dan mempelajari karakteristik objek telur berdasarkan dua kelas, yaitu Telur Utuh dan Telur Retak. Pada penelitian ini, jumlah data yang digunakan untuk pelatihan sebanyak 734 citra.

2. Data Validasi

Data validasi digunakan untuk melakukan evaluasi performa model secara sementara selama proses pelatihan berlangsung. *Dataset* ini berfungsi untuk memantau nilai akurasi dan *loss* pada setiap *epoch*, serta mencegah terjadinya *overfitting* atau *underfitting*. Pada penelitian ini, data validasi yang digunakan sebanyak 150 citra.

3. Data *Testing*

Data *testing* digunakan untuk menguji kinerja akhir model setelah proses pelatihan selesai. Data ini bersifat independen, karena tidak dilibatkan dalam proses pelatihan maupun validasi. Pada penelitian ini, data *testing* digunakan untuk mengukur kemampuan model dalam mendeteksi dan mengklasifikasikan kondisi telur secara akurat. Jumlah data yang digunakan untuk pengujian sebanyak 75 citra.

**Gambar 13. Contoh Data Testing**

d. Augmentasi Data Latih

Augmentasi merupakan teknik pengolahan citra dan *machine learning* yang digunakan untuk memperbanyak data pelatihan dengan memodifikasi citra asli (Maharana et al., 2022). Pada penelitian, ini Augmentasi hanya dilakukan pada data *training* karena tujuannya adalah memperkaya variasi data agar model dapat belajar lebih baik dan menjadi lebih tangguh terhadap berbagai kondisi *input*. Sebaliknya, data validasi dan *testing* harus tetap dalam kondisi asli agar dapat memberikan evaluasi yang objektif dan akurat terhadap performa model pada data nyata yang belum pernah dilihat sebelumnya. Data *training* pada penelitian ini dilakukan teknik augmentasi *rotation*, *flip*. Data latih yang dihasilkan setelah melalui proses augmentasi sebanyak 734 gambar yang nantinya siap digunakan untuk melatih model.

### 2.4.2 *Training*

1. Input Data Latih

Setelah melalui pre prosesing data, data *training* yang berjumlah 734 digunakan untuk melatih model deteksi. Data *training* ini menjadi dasar bagi model dalam menyesuaikan bobot atau *weight*, sehingga mampu melakukan deteksi dan klasifikasi secara akurat terhadap citra telur ayam yang belum pernah dilihat sebelumnya.

2. Training Model yolov8 dengan DCNv2

YOLOv8 merupakan algoritma deteksi objek generasi terbaru yang dikembangkan oleh Ultralytics, dengan arsitektur utama terdiri atas tiga komponen, yaitu *backbone*, *neck*, dan *head*. Komponen *backbone* berfungsi mengekstraksi fitur dari citra masukan melalui lapisan konvolusi dan blok C2f (*Cross Stage* Partial *with* 2 *Convolutions*), yang dirancang untuk meningkatkan efisiensi komputasi tanpa mengurangi kedalaman representasi fitur. Pada bagian akhir *backbone*, modul SPPF (*Spatial Pyramid Pooling Fast*) digunakan untuk memperluas *receptive field* dan menangkap konteks spasial yang lebih luas (Zhao, 2024). Selanjutnya, bagian *neck* mengimplementasikan kombinasi FPN-PAN (*Feature Pyramid Network Path Aggregation Network*) yang berperan menggabungkan fitur dari berbagai tingkat resolusi, sehingga informasi detail dari level rendah dan informasi semantik dari level tinggi dapat terintegrasi secara optimal.

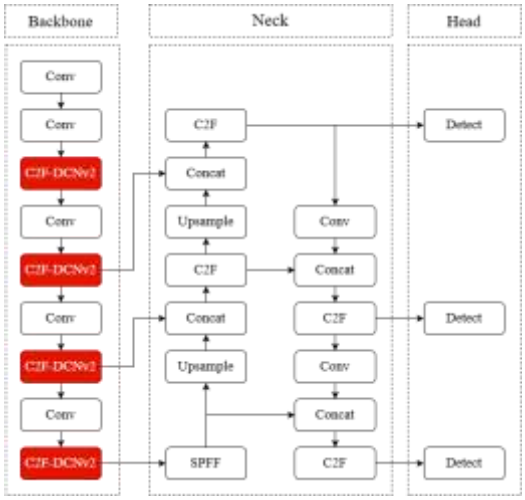
Bagian *head* terdiri atas lapisan deteksi multi skala yang menghasilkan prediksi *bounding box*, *confidence score*, serta klasifikasi objek pada berbagai ukuran. Meskipun arsitektur YOLOv8 dikenal memiliki kecepatan inferensi tinggi dan akurasi yang andal, model *baseline* ini masih menghadapi keterbatasan dalam mendeteksi pola objek yang halus dan tidak beraturan, seperti retakan tipis pada permukaan cangkang telur, akibat sifat konvolusi standar yang menggunakan *kernel* tetap (*rigid kernel*) sehingga kurang adaptif terhadap variasi bentuk dan deformasi objek.

Arsitektur YOLOv8 perlu dimodifikasi karena model asli (*baseline*) masih memiliki kelemahan dalam mendeteksi retakan telur yang sangat tipis, kecil, dan

bentuknya tidak beraturan. Pada YOLOv8 standar, proses ekstraksi fitur menggunakan konvolusi biasa dengan posisi kernel yang tetap. Kondisi ini membuat model kurang fleksibel saat membaca pola retakan yang memiliki bentuk melengkung, bercabang, atau berubah-ubah pada permukaan cangkang telur. Akibatnya, beberapa retakan sulit dikenali sehingga dapat menyebabkan kesalahan deteksi, seperti retakan tidak terdeteksi (*false negative*) atau bagian bukan retakan dianggap retakan (*false positive*).

Untuk mengatasi masalah tersebut, dilakukan modifikasi arsitektur dengan menambahkan metode Deformable Convolutional Networks v2 (DCNv2) pada YOLOv8. Modifikasi ini bertujuan agar proses ekstraksi fitur menjadi lebih adaptif terhadap bentuk objek. Berbeda dengan konvolusi standar yang posisi kernelnya tetap, DCNv2 mampu menggeser titik sampling secara dinamis mengikuti pola dan kontur objek. Dengan cara ini, model dapat lebih fokus menangkap detail retakan yang tipis dan tidak beraturan.

Melalui modifikasi tersebut, model diharapkan mampu mengenali pola retakan dengan lebih akurat dibandingkan YOLOv8 sebelumnya. Selain meningkatkan kemampuan deteksi retakan kecil, integrasi DCNv2 juga membantu model menghasilkan klasifikasi yang lebih stabil dan konsisten pada berbagai kondisi permukaan telur. Oleh karena itu, performa model hasil modifikasi menjadi lebih baik karena fitur penting pada objek dapat diekstraksi secara lebih optimal dibandingkan model baseline yang masih menggunakan konvolusi standar.




**Gambar 14. Arsitektur YOLOv8+DCNv2**

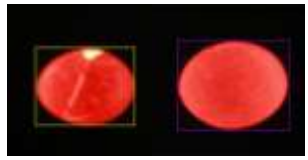
### a. Inisiasi Bobot awal

Sebelum proses pelatihan dimulai, bobot awal pada jaringan konvolusional di inisialisasi secara acak. Inisialisasi ini dilakukan untuk memberikan nilai awal pada parameter model agar proses optimasi dapat berjalan. Pemilihan bobot secara *random* memungkinkan model memulai pembelajaran tanpa bias terhadap kelas atau fitur tertentu. Setelah inisialisasi, bobot akan diperbarui secara bertahap selama proses pelatihan berdasarkan *loss* yang dihasilkan dari prediksi model

### b. *Input* data *training*

Pada tahap awal dalam proses pelatihan model deteksi objek seperti YOLO, digunakan *dataset* yang terdiri dari citra objek sebagai *input*. Setiap gambar ini

dilengkapi dengan *ground truth*, yaitu anotasi yang menjelaskan posisi objek melalui koordinat *bounding box* serta label kelasnya, seperti utuh dan retak. Setiap kelas dilakukan *encoding* untuk diubah menjadi angka, di mana kelas utuh menjadi angka 1 dan kelas retak diubah menjadi angka 0. Gambar 15 contoh citra yang digunakan dalam proses *training* yang telah di anotasi.



**Gambar 15. Contoh *Input* Citra Proses *Training***

Anotasi menjadi acuan utama bagi model untuk belajar mengenali pola visual dan mengaitkan dengan lokasi serta kategori objek yang sesuai dalam gambar. *Ground truth* inilah yang menjadi target pembelajaran untuk meminimalkan kesalahan prediksi selama *training*. Gambar 16 merupakan contoh anotasi inputan *training*.

| | file_name | category | x | y | width | height |
|---|---|---|---|---|---|---|
| 0 | 002-144-_jpg.rf.2e28896c7c55b40a613c597b1a0b76... | 1 | 228 | 524 | 711.1 | 490.4 |
| 1 | 002-145-_jpg.rf.43e65f38d83b43242b6425a7929d12... | 1 | 815 | 537 | 686.7 | 478.7 |
| 2 | 002-146-_jpg.rf.0323ee3a2aaf86fd95f1f304abf66c... | 1 | 728 | 542 | 690.0 | 479.3 |
| 3 | 002-147-_jpg.rf.5beee8b1c822782f34eabfd8a248ef... | 1 | 922 | 537 | 711.1 | 492.1 |
| 4 | 002-157-_jpg.rf.324a959346d098a6c7026ee0c26589... | 1 | 902 | 529 | 691.1 | 487.6 |
| ... | ... | ... | ... | ... | ... | ... |
| 729 | frame_3605_jpg.rf.7c8759ffe7f3482164e27d90f8327... | 0 | 1103 | 584 | 661.5 | 457.9 |
| 730 | frame_3648_jpg.rf.b4336f21033952a02b0f5a254469... | 0 | 807 | 510 | 593.7 | 448.2 |
| 731 | frame_3649_jpg.rf.a096492c5300dd8b35f0eb47b1b0... | 0 | 961 | 516 | 593.7 | 448.2 |
| 732 | frame_3650_jpg.rf.60bb295256874c3fabbc0ca441ec... | 0 | 1033 | 516 | 597.7 | 448.2 |
| 733 | frame_3651_jpg.rf.0ede930c5229e9e9a7ddb8706a08... | 0 | 1307 | 525 | 597.7 | 448.2 |

734 rows × 6 columns

**Gambar 16. File Anotasi *Input Training***

**Gambar 17. data inputan**

Pada gambar di atas, setiap titik pada citra disebut piksel. Setiap piksel memiliki nilai intensitas warna yang tersusun dari channel *Red* (R), *Green* (G), dan *Blue* (B). Karena gambar berbentuk RGB, maka setiap piksel memiliki tiga nilai warna dengan rentang 0–255. Saat disebut mengambil nilai piksel 5×5, artinya hanya diambil sebagian kecil area citra yang terdiri dari 5 baris dan 5 kolom piksel, sehingga total terdapat 25 piksel yang diamati. Proses ini biasanya dilakukan untuk mempermudah analisis awal citra, melihat pola intensitas warna, atau menjelaskan cara kerja konvolusi pada CNN.

Pada tahap akuisisi, citra telur diperoleh menggunakan kamera dengan resolusi tertentu sehingga menghasilkan data digital berupa kumpulan piksel. Setiap piksel pada citra memiliki nilai intensitas yang merepresentasikan tingkat kecerahan

atau warna pada bagian tertentu dari objek telur. Setelah citra diperoleh, dilakukan penyesuaian ukuran citra menjadi $640 \times 640 \times 3$ agar sesuai dengan input model YOLOv8. Selanjutnya, sistem memilih sebagian kecil area citra sebagai sampel untuk mempermudah proses penjelasan perhitungan normalisasi. Area sampel dipilih dari bagian tengah objek telur karena bagian tersebut dianggap mampu mewakili karakteristik citra secara umum dan mengurangi pengaruh *noise* atau latar belakang. Sampel tersebut diambil dalam ukuran $5 \times 5$, sehingga menghasilkan 25 nilai piksel yang kemudian direpresentasikan dalam bentuk matriks sebagai berikut:

$$\begin{bmatrix} 12 & 45 & 67 & 23 & 89 \\ 54 & 31 & 78 & 16 & 42 \\ 90 & 25 & 8 & 64 & 37 \\ 48 & 72 & 19 & 95 & 6 \\ 33 & 81 & 14 & 58 & 27 \end{bmatrix}$$

Nilai-nilai pada matriks tersebut merupakan intensitas piksel yang masih berada pada rentang asli citra digital, yaitu 0–255. Nilai 0 menunjukkan warna paling gelap (hitam), sedangkan nilai 255 menunjukkan warna paling terang (putih). Semakin besar nilai piksel, maka semakin terang intensitas cahaya pada area tersebut.
Namun, nilai piksel yang terlalu besar dapat menyebabkan proses pelatihan model menjadi kurang stabil dan memperlambat proses pembelajaran. Oleh karena itu, dilakukan proses normalisasi agar seluruh nilai piksel berada pada rentang 0–1. Dengan nilai yang lebih kecil dan seragam, model CNN atau YOLO dapat lebih mudah mempelajari pola citra, mempercepat konvergensi pelatihan, dan meningkatkan stabilitas komputasi.
Metode normalisasi yang digunakan adalah *Min-Max Normalization*. Metode ini bekerja dengan cara mengubah setiap nilai piksel berdasarkan nilai minimum dan maksimum pada data. Rumus normalisasi ditunjukkan sebagai berikut:

$$x' = \frac{x - x_{min}}{x_{max} - x_{min}} \tag{6}$$

Keterangan:

$x$= nilai asli piksel
$x_{min}$= nilai minimum pada data
$x_{max}$= nilai maksimum pada data
$x'$= hasil normalisasi

Berdasarkan matriks piksel di atas, diperoleh:

$$x_{min} = 6$$
$$x_{max} = 95$$

Artinya, nilai piksel terkecil pada matriks adalah 6, sedangkan nilai terbesar adalah 95. Selanjutnya nilai tersebut dimasukkan ke dalam rumus normalisasi:

$$x' = \frac{x - 6}{95 - 6}$$
$$x' = \frac{x - 6}{89}$$

Rumus tersebut digunakan untuk mengubah seluruh nilai piksel agar berada pada rentang 0–1. Sebagai contoh, dilakukan perhitungan pada beberapa nilai piksel berikut.

1. Normalisasi nilai piksel 12

$$x' = \frac{12 - 6}{89}$$
$$x' = \frac{6}{89}$$
$$x' = 0.067$$

Hasil tersebut menunjukkan bahwa nilai piksel 12 berubah menjadi 0.067 setelah dinormalisasi.

2. Normalisasi nilai piksel 45

$$x' = \frac{45 - 6}{89}$$
$$x' = \frac{39}{89}$$
$$x' = 0.438$$

Artinya, nilai piksel 45 berubah menjadi 0.438.

3. Normalisasi nilai piksel 95

$$x' = \frac{95 - 6}{89}$$
$$x' = \frac{89}{89}$$
$$x' = 1.000$$

Karena 95 merupakan nilai maksimum pada matriks, maka hasil normalisasinya menjadi 1. Setelah seluruh nilai dihitung menggunakan rumus normalisasi, diperoleh matriks hasil normalisasi sebagai berikut:

$$\begin{bmatrix} 0.067 & 0.438 & 0.685 & 0.191 & 0.933 \\ 0.539 & 0.281 & 0.809 & 0.112 & 0.404 \\ 0.944 & 0.213 & 0.022 & 0.652 & 0.348 \\ 0.472 & 0.742 & 0.146 & 1.000 & 0.000 \\ 0.303 & 0.843 & 0.090 & 0.584 & 0.236 \end{bmatrix}$$

Dapat dilihat bahwa seluruh nilai piksel sekarang berada pada rentang 0 sampai 1. Nilai yang mendekati 0 menunjukkan intensitas gelap, sedangkan nilai yang mendekati 1 menunjukkan intensitas terang.

Setelah proses normalisasi selesai, dilakukan proses pengembalian nilai ke rentang 0–255 untuk keperluan visualisasi citra. Proses ini disebut *denormalisasi*. Tujuannya adalah agar hasil citra tetap dapat ditampilkan kembali dalam format citra digital yang umum digunakan komputer.

Rumus denormalisasi adalah:

$$x'' = x' \times 255 \quad (7)$$

Keterangan:

$x''$= nilai piksel hasil visualisasi

$x'$= nilai hasil normalisasi

Contoh perhitungan:

Untuk nilai 0.067:

$$0.067 \times 255 = 17.2$$

Untuk nilai 0.438:

$$0.438 \times 255 = 111.7$$

Untuk nilai 1.000:

$$1.000 \times 255 = 255$$

Setelah seluruh nilai dikembalikan ke rentang 0–255, diperoleh matriks visualisasi sebagai berikut:

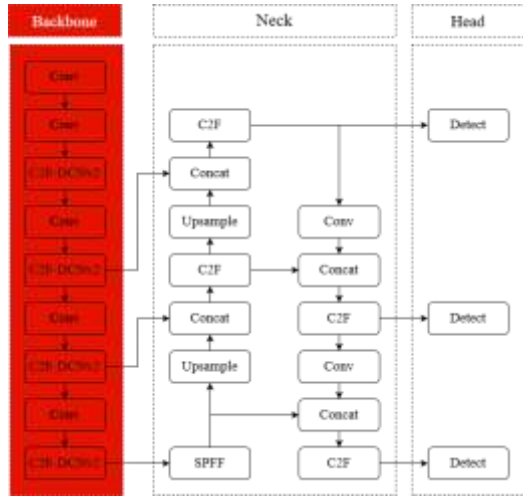
$$\begin{bmatrix} 17.2 & 111.7 & 174.8 & 48.7 & 237.8 \\ 137.5 & 71.6 & 206.3 & 28.7 & 103.2 \\ 240.7 & 54.4 & 5.7 & 166.2 & 88.8 \\ 120.3 & 189.1 & 37.3 & 255.0 & 0.0 \\ 77.4 & 214.9 & 22.9 & 150.0 & 60.2 \end{bmatrix}$$

Berdasarkan hasil tersebut, dapat dipahami bahwa proses normalisasi berfungsi untuk memperkecil rentang nilai piksel agar proses komputasi pada CNN atau YOLO menjadi lebih stabil dan efisien. Setelah diproses, nilai tersebut dapat dikembalikan

lagi ke bentuk semula untuk divisualisasikan sebagai citra digital. Dengan demikian, tahap normalisasi menjadi langkah penting dalam meningkatkan kualitas proses pembelajaran model deteksi keretakan telur.

### c. Tahapan *Backbone*

*Backbone* merupakan bagian utama dari arsitektur model yang bertugas melakukan ekstraksi fitur secara bertahap dari citra *input*. Pada tahap ini, setiap blok seperti *Conv* dan C2F-DCNv2 memiliki peran spesifik dalam memperkuat informasi visual, mulai dari pola sederhana hingga struktur yang lebih kompleks. Urutan lapisan pada *backbone* dirancang untuk menangkap tekstur, bentuk, dan deformasi permukaan secara progresif, sehingga menghasilkan representasi fitur yang kaya dan stabil sebelum diteruskan ke tahap *neck* dan *head*. Dengan memahami fungsi setiap komponen *backbone*, kita dapat melihat bagaimana model membangun pemahaman visual yang menjadi dasar akurasi deteksi. Berikut ini Gambar 18 bagian *backbone* sebagai berikut ini :




**Gambar 18. Tahapan *Backbone***

#### 1. Nilai *patch input* (3×3 per *channel*)

Pada tahap awal proses ekstraksi fitur, citra masukan terlebih dahulu dibagi menjadi bagian-bagian kecil yang disebut *patch*. Patch merupakan area lokal kecil dari citra yang digunakan untuk membantu model membaca pola tertentu secara bertahap. Pada penelitian ini digunakan patch berukuran $3 \times 3$, artinya terdapat 3 baris dan 3 kolom piksel pada setiap area yang diproses.

Karena citra digital berwarna terdiri atas tiga kanal warna RGB (*Red, Green, Blue*), maka setiap patch sebenarnya memiliki tiga lapisan data warna. Setiap kanal menyimpan informasi intensitas warna yang berbeda:

a) Kanal *Red* (R) menyimpan intensitas warna merah.
b) Kanal *Green* (G) menyimpan intensitas warna hijau.
c) Kanal *Blue* (B) menyimpan intensitas warna biru.

Namun, untuk mempermudah penjelasan proses konvolusi, ilustrasi berikut menggunakan satu patch sederhana berukuran $3 \times 3$yang mewakili salah satu kanal warna atau hasil citra grayscale.

Patch citra yang digunakan adalah:

$$\begin{bmatrix} 17.2 & 111.7 & 174.8 \\ 137.5 & 71.6 & 206.3 \\ 240.7 & 54.4 & 5.7 \end{bmatrix}$$

Setiap angka pada matriks tersebut merupakan nilai intensitas piksel. Nilai kecil menunjukkan area gelap, sedangkan nilai besar menunjukkan area terang pada citra telur hasil *candling*.

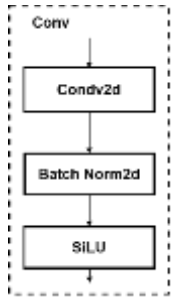
Patch kecil ini kemudian diproses oleh modul *Convolution* untuk mengekstraksi fitur penting seperti garis retakan, tekstur cangkang, pola cahaya, dan bentuk objek telur.

## 2. Modul *Convolution* (*Conv*)

Modul *Convolution* (*Conv*) pada YOLOv8 merupakan bagian utama dalam proses ekstraksi fitur citra. Modul ini bekerja secara berurutan (*sequential*) melalui tiga tahap utama, yaitu:

a) Conv2D (*Convolution* 2 Dimensional).
b) *Batch Normalization* (BN).
c) Fungsi Aktivasi SiLU (*Sigmoid Linear Unit*).

Tahapan tersebut bekerja bersama untuk menghasilkan fitur yang lebih informatif sebelum diteruskan ke lapisan berikutnya pada jaringan YOLOv8.




**Gambar 19. Alur *Convolution* (*Conv*)**

**Conv2d:** *Convolution* 2 Dimensi adalah operasi matematika pada CNN (*Convolutional Neural Network*) yang digunakan untuk mengekstraksi fitur penting dari citra. Pada proses ini digunakan sebuah filter kecil yang disebut *kernel* atau *filter convolution*.

*Kernel* akan digeser di atas area citra sambil melakukan operasi perkalian dan penjumlahan antara nilai piksel citra dan nilai kernel. Hasil perhitungan tersebut menghasilkan nilai baru yang disebut *feature map*.

Feature map berisi informasi penting dari citra seperti:

a) Tepi objek
b) Tekstur permukaan
c) Pola retakan
d) Bentuk objek
e) Perbedaan intensitas cahaya

Pada penelitian deteksi keretakan telur, Conv2D membantu model mengenali pola retakan kecil pada cangkang telur hasil pencahayaan *candling*.

Pada citra digital, Conv2d biasanya digunakan untuk mendeteksi pola tertentu. Misalnya pada penelitian deteksi retakan telur, Conv2d membantu model mengenali garis retakan, tekstur cangkang, dan perbedaan intensitas cahaya pada telur hasil candling.

Sebagai contoh sederhana, misalkan terdapat patch citra 3×3:

$$\begin{bmatrix} 17.2 & 111.7 & 174.8 \\ 137.5 & 71.6 & 206.3 \\ 240.7 & 54.4 & 5.7 \end{bmatrix}$$

Misalkan digunakan kernel berukuran $3 \times 3$:

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Kernel di atas disebut *mean filter sederhana* karena seluruh elemennya bernilai 1. Fungsi kernel ini adalah membaca informasi intensitas dari area citra yang sedang diproses. Berikut ini Langkah Perhitungan:

Baris pertama

$$(17.2 \times 1) + (111.7 \times 1) + (174.8 \times 1)$$
$$= 17.2 + 111.7 + 174.8$$
$$= 303.7$$

Baris kedua

$$(137.5 \times 1) + (71.6 \times 1) + (206.3 \times 1)$$
$$= 137.5 + 71.6 + 206.3$$
$$= 415.4$$

Baris ketiga

$$(240.7 \times 1) + (54.4 \times 1) + (5.7 \times 1)$$
$$= 240.7 + 54.4 + 5.7$$
$$= 300.8$$

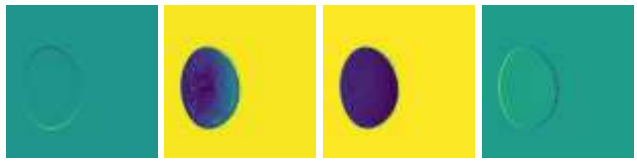
Total Hasil Konvolusi Seluruh hasil kemudian dijumlahkan:

$$303.7 + 415.4 + 300.8$$
$$= 1019.9$$

Maka diperoleh nilai output konvolusi:

$$1019.9$$

Nilai tersebut menjadi satu nilai baru pada *feature map* hasil konvolusi. Nilai 1019.9 tersebut menjadi satu nilai output pada feature map hasil konvolusi. Setelah itu, kernel akan bergeser ke area lain pada citra untuk menghasilkan nilai output berikutnya hingga terbentuk feature map baru. berikut ini contoh dari hasil Conv2D:



**Gambar 20. Hasil Conv2D**

BN: *Batch Normalization* (BN) Setelah proses Conv2D selesai dilakukan, tahap berikutnya pada modul convolution YOLOv8 adalah *Batch Normalization* (*BatchNorm2d*). Tahap ini berfungsi untuk menstabilkan distribusi data hasil konvolusi agar proses pelatihan (*training*) menjadi lebih cepat, stabil, dan model lebih mudah mempelajari pola pada citra. Pada CNN, hasil konvolusi sering menghasilkan nilai feature map yang terlalu besar atau memiliki distribusi yang tidak stabil. Jika kondisi ini dibiarkan, proses pembelajaran model dapat menjadi lambat dan berisiko mengalami *overfitting* maupun *vanishing gradient*. Oleh karena itu, Batch Normalization digunakan untuk menormalkan nilai feature map sehingga data memiliki distribusi yang lebih seimbang. Output feature map hasil Conv2D yang digunakan pada contoh ini adalah:

$$\begin{bmatrix} 1019.9 & 868.1 & 1060.2 \\ 1062.9 & 1014.3 & 891.2 \\ 962.7 & 1099.5 & 786.1 \end{bmatrix}$$

Feature map tersebut terdiri dari 9 nilai hasil ekstraksi fitur yang diperoleh dari proses konvolusi sebelumnya.

a. Menghitung Mean $(\mu)$

Langkah pertama pada Batch Normalization adalah menghitung nilai rata-rata (*mean*) dari seluruh data pada feature map.

Rumus mean:

$$\mu = \frac{\sum x}{n} \tag{8}$$

Keterangan:

$\mu$= mean atau rata-rata data

$\sum x$= jumlah seluruh nilai data

$n$= jumlah data

Jumlah seluruh nilai feature map:

$$1019.9 + 868.1 + 1060.2 + 1062.9 + 1014.3 + 891.2 + 962.7 + 1099.5 + 786.1$$
$$= 8765.0$$

Karena jumlah data sebanyak 9, maka:

$$\mu = \frac{8765.0}{9}$$
$$\mu = 973.9$$

Nilai mean sebesar 973.9 menunjukkan rata-rata intensitas *feature map* hasil konvolusi

b. Menghitung *Variansi* $(\sigma^2)$

Setelah *mean* diperoleh, langkah berikutnya adalah menghitung *variansi*. Variansi digunakan untuk mengetahui seberapa jauh penyebaran data terhadap nilai rata-rata. Rumus variansi:

$$\sigma^2 = \frac{\sum (x - \mu)^2}{n} \tag{9}$$

Keterangan:

$\sigma^2$= *variansi*

$x$= nilai data

$\mu$= *mean*

$n$= jumlah data

Perhitungan Variansi Untuk nilai pertama: 1019.9 Langkah pertama adalah menghitung selisih terhadap mean:

$$x - \mu = 1019.9 - 973.9$$
$$= 46.0$$

Kemudian hasil tersebut dikuadratkan:

$$(46.0)^2$$
$$= 2116.0$$

Untuk nilai kedua: 868.1

$$868.1 - 973.9 = -105.8$$
$$(-105.8)^2 = 11193.6$$

Untuk nilai ketiga: 1060.2

$$1060.2 - 973.9 = 86.3$$
$$(86.3)^2 = 7447.7$$

Perhitungan tersebut dilakukan pada seluruh nilai *feature map* hingga diperoleh total:

$$\sum(x-\mu)^2 = 88319.4$$

Maka nilai *variansi*:

$$\sigma^2 = \frac{88319.4}{9}$$
$$\sigma^2 = 9813.3$$

*Variansi* sebesar 9813.3 menunjukkan tingkat penyebaran data terhadap nilai rata-rata.

c. Proses *Batch Normalization*

Setelah *mean* dan *variansi* diperoleh, langkah berikutnya adalah melakukan normalisasi terhadap setiap nilai *feature map* menggunakan rumus *Batch Normalization*:

$$y = \gamma \cdot \frac{x-\mu}{\sqrt{\sigma^2+\epsilon}} + \beta \quad (10)$$

Dengan:

$\gamma = 1$
$\beta = 0$
$\epsilon = 0.00001$

Perhitungan B*atchNorm*

Untuk nilai 1019.9 *Substitusi* ke rumus:

$$y = 1 \cdot \frac{1019.9 - 973.9}{\sqrt{9813.3 + 0.00001}} + 0$$

Hitung pembilang:

$$1019.9 - 973.9 = 46.0$$

Hitung penyebut:

$$\sqrt{9813.3 + 0.00001}$$
$$= 99.06$$

Sehingga:

$$y = \frac{46.0}{99.06}$$
$$y = 0.46$$

Hasil tersebut menunjukkan bahwa nilai 1019.9 berada sedikit di atas rata-rata data.

nilai kedua: 868.1

$$y = \frac{868.1 - 973.9}{99.06}$$
$$y = \frac{-105.8}{99.06}$$
$$y = -1.07$$

Nilai negatif menunjukkan data berada di bawah rata-rata.

Hasil Output *BatchNorm* Setelah seluruh nilai dihitung menggunakan rumus *Batch Normalization*, diperoleh output sebagai berikut:

$$\begin{bmatrix} 0.46 & -1.07 & 0.87 \\ 0.90 & 0.41 & -0.83 \\ -0.11 & 1.27 & -1.89 \end{bmatrix}$$

Interpretasi Hasil *Batch Normalization* menghasilkan data dengan distribusi yang lebih stabil dan terpusat di sekitar nol.

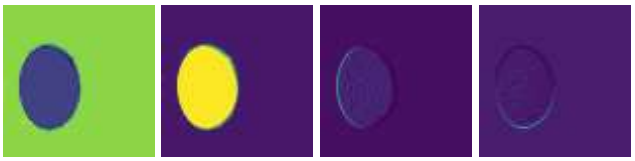
Interpretasi nilai:

a) Nilai positif menunjukkan fitur berada di atas rata-rata
b) Nilai negatif menunjukkan fitur berada di bawah rata-rata
c) Nilai mendekati nol menunjukkan fitur berada dekat rata-rata

Sebagai contoh:

d) Nilai 1.27 menunjukkan fitur cukup tinggi dibanding rata-rata
e) Nilai -1.89 menunjukkan fitur jauh lebih rendah dibanding rata-rata
f) Nilai 0.41 menunjukkan fitur sedikit di atas rata-rata

Nilai positif menunjukkan fitur berada di atas rata-rata, sedangkan nilai negatif menunjukkan fitur berada di bawah rata-rata. Berikut merupakan visualisasi keluaran *Batch Normalization* (BN):



**Gambar 21. Hasil Dari *Batch Normalization* (BN)**

Hasil visualisasi *Batch Normalization* (BN) pada Gambar 20 menunjukkan nilai aktivasi yang telah di normalisasi menggunakan parameter *running mean*, *running variance*, serta faktor skala dan pergeseran. Proses ini bertujuan menstabilkan distribusi data sehingga keluaran menjadi lebih konsisten dan siap diproses oleh lapisan berikutnya dalam jaringan saraf.

**Aktivasi SiLU**: Setelah proses Batch Normalization selesai dilakukan, tahap berikutnya pada modul convolution YOLOv8 adalah fungsi aktivasi SiLU (*Sigmoid Linear Unit)*. Fungsi aktivasi ini digunakan untuk memberikan sifat *non-linear* pada jaringan saraf (*neural network*) sehingga model mampu mempelajari pola yang lebih kompleks pada citra. Tanpa fungsi aktivasi, jaringan CNN hanya akan melakukan operasi *linear* sederhana sehingga model sulit membedakan pola kompleks seperti:

a) Retakan tipis pada telur
b) Tekstur cangkang
c) Perbedaan intensitas cahaya
d) Pola garis halus hasil *candling*
e) Oleh karena itu, fungsi aktivasi menjadi bagian penting dalam proses pembelajaran model YOLOv8.

Fungsi Aktivasi SiLU bekerja dengan mengalikan nilai input dengan hasil fungsi sigmoid dari input tersebut Rumus SiLU:

$$\mathrm{SiLU}(y) = y \cdot \sigma(y) \tag{11}$$

Dengan fungsi sigmoid:

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

Keterangan:

$y$= nilai input dari *Batch Normalization*
$\sigma(y)$= hasil fungsi *sigmoid*
$\mathrm{SiLU}(y)$= output akhir fungsi aktivasi

Output *Batch Normalization* Nilai input yang digunakan berasal dari hasil *BatchNorm* sebelumnya:

$$\begin{bmatrix} 0.46 & -1.07 & 0.87 \\ 0.90 & 0.41 & -0.83 \\ -0.11 & 1.27 & -1.89 \end{bmatrix}$$

Nilai tersebut kemudian diproses menggunakan fungsi *sigmoid* dan SiLU.

a. Menghitung Nilai *Sigmoid*

Tahap pertama adalah menghitung nilai sigmoid untuk setiap elemen matriks.

Fungsi sigmoid mengubah nilai input menjadi rentang 0–1. Nilai ini berfungsi sebagai pengontrol seberapa besar informasi dari input akan diteruskan ke lapisan berikutnya. Perhitungan *Sigmoid*

1. Untuk nilai $y = 0.46$

Substitusi ke rumus *sigmoid*:

$$\sigma(0.46) = \frac{1}{1 + e^{-0.46}}$$

Hitung nilai *eksponensial*:

$$e^{-0.46} \approx 0.63$$

Sehingga:

$$\sigma(0.46) = \frac{1}{1 + 0.63}$$
$$= \frac{1}{1.63}$$
$$= 0.61$$

Artinya, nilai 0.46 memiliki tingkat aktivasi sebesar 0.61.

2. Untuk nilai $y = -1.07$

$$\sigma(-1.07) = \frac{1}{1 + e^{1.07}}$$

Karena:

$$e^{1.07} \approx 2.91$$

Maka:

$$\sigma(-1.07) = \frac{1}{1 + 2.91}$$
$$= \frac{1}{3.91}$$
$$= 0.26$$

Nilai negatif menghasilkan nilai *sigmoid* yang kecil sehingga informasi yang diteruskan juga lebih kecil.

3. Untuk nilai $y = 0.87$

$$\sigma(0.87) = \frac{1}{1 + e^{-0.87}}$$
$$e^{-0.87} \approx 0.42$$
$$\sigma(0.87) = \frac{1}{1 + 0.42}$$
$$= \frac{1}{1.42}$$
$$= 0.70$$

Hasil Seluruh Matriks *Sigmoid* Setelah semua nilai dihitung, diperoleh hasil *sigmoid* sebagai berikut:

$$\begin{bmatrix} 0.61 & 0.26 & 0.70 \\ 0.71 & 0.60 & 0.30 \\ 0.47 & 0.78 & 0.13 \end{bmatrix}$$

Nilai sigmoid selalu berada pada rentang 0–1:
a) Nilai mendekati 1 menunjukkan aktivasi kuat
b) Nilai mendekati 0 menunjukkan aktivasi lemah

b. Menghitung Aktivasi SiLU

Setelah nilai sigmoid diperoleh, langkah berikutnya adalah menghitung output SiLU. Artinya:
a) Nilai input dikalikan dengan hasil sigmoidnya sendiri
b) Informasi penting tetap dipertahankan
c) Nilai kecil atau negatif akan direduksi secara halus

Perhitungan SiLU

1. Untuk nilai $y = 0.46$

Diketahui:

$$\sigma(0.46) = 0.61$$

Maka:

$$\text{SiLU}(0.46) = 0.46 \times 0.61$$
$$= 0.2$$

2. Untuk nilai $y = -1.07$

Diketahui:

$$\sigma(-1.07) = 0.26$$

Maka:

$$\text{SiLU}(-1.07) = -1.07 \times 0.26$$
$$= -0.28$$

Nilai negatif tetap dipertahankan tetapi diredam secara halus.

3. Untuk nilai $y = 0.87$

Diketahui:

$$\sigma(0.87) = 0.70$$

Maka:

$$\text{SiLU}(0.87) = 0.87 \times 0.70$$
$$= 0.6$$

Hasil Akhir Aktivasi SiLU Setelah seluruh nilai dihitung menggunakan fungsi SiLU, diperoleh output akhir sebagai berikut:

$$\begin{bmatrix} 0.28 & -0.28 & 0.61 \\ 0.64 & 0.25 & -0.25 \\ -0.05 & 0.99 & -0.25 \end{bmatrix}$$

Interpretasi Hasil SiLU Hasil aktivasi SiLU menunjukkan bagaimana jaringan memilih informasi penting dari feature map Interpretasi nilai:
a) Nilai positif besar tetap dipertahankan karena dianggap penting
b) Nilai negatif tidak langsung dibuang, tetapi direduksi secara halus
c) Nilai kecil akan memiliki pengaruh lebih kecil pada proses berikutnya

Sebagai contoh:
a) Nilai 0.99 menunjukkan fitur sangat kuat dan penting
b) Nilai 0.28 menunjukkan fitur cukup aktif
c) Nilai -0.25 menunjukkan fitur kurang penting tetapi masih dipertahankan sebagian

Hasil ini menunjukkan bahwa untuk nilai masukan positif besar, fungsi SiLU mempertahankan hampir seluruh nilai masukan dengan sedikit peredaman non linier, sehingga tetap menjaga informasi penting dari fitur yang diekstraksi.

berikut ini Gambar 22 hasil dari proses SiLU:

**Gambar 22. Hasil Dari SiLU**

Selain berfungsi untuk mengekstraksi fitur dari citra, operasi **Conv2D** juga menyebabkan perubahan dimensi pada *feature map*. Perubahan ukuran ini dipengaruhi oleh beberapa parameter utama, yaitu ukuran kernel (*kernel size*), *stride*, dan *padding*. Ketiga parameter tersebut menentukan bagaimana kernel bergerak pada citra dan berapa ukuran output yang dihasilkan setelah proses konvolusi selesai dilakukan.

Pada arsitektur YOLOv8, proses konvolusi umumnya menggunakan nilai *stride* tertentu untuk mengurangi ukuran spasial citra secara bertahap. *Stride* merupakan jumlah langkah pergeseran kernel saat membaca area citra. Jika *stride* bernilai 1, maka kernel bergeser satu piksel setiap proses konvolusi. Sedangkan jika *stride* bernilai 2, kernel akan bergeser dua piksel sehingga ukuran output menjadi lebih kecil. Sebagai contoh, jika citra masukan memiliki ukuran:

$$640 \times 640$$

dan proses Conv2D menggunakan:

a) kernel $3 \times 3$
b) stride $= 2$
c) padding tertentu

maka ukuran output akan berkurang menjadi:

$$320 \times 320$$

Artinya, panjang dan lebar feature map menjadi setengah dari ukuran sebelumnya. Pengurangan ukuran ini disebut proses *downsampling*. Ilustrasi perubahan ukuran feature map:

$$640 \times 640 \rightarrow 320 \times 320 \rightarrow 160 \times 160 \rightarrow 80 \times 80$$

Semakin dalam jaringan CNN, ukuran spasial feature map akan semakin kecil, tetapi jumlah channel biasanya semakin besar. Tujuan pengurangan ukuran feature map adalah:

a) Mengurangi beban komputasi
b) Mempercepat proses training dan inferensi
c) Mengurangi penggunaan memori
d) Membantu model fokus pada fitur penting
e) Memperluas *receptive field* agar model memahami konteks objek secara lebih luas

Dalam penelitian deteksi keretakan telur, proses ini membantu model mengenali pola retakan secara lebih efisien tanpa harus memproses seluruh detail piksel berukuran besar.

Selain ukuran spasial, Conv2D juga mengubah jumlah channel pada feature map. Jumlah channel output ditentukan oleh jumlah kernel (*filter*) yang digunakan pada proses konvolusi. Sebagai contoh:

a) Input awal RGB memiliki 3 channel
b) Jika Conv2D menggunakan 64 kernel

maka output feature map akan memiliki:

$$320 \times 320 \times 64$$

Artinya:
a) ukuran feature map menjadi $320 \times 320$
b) terdapat 64 channel feature map hasil ekstraksi
c) Setiap kernel memiliki tugas mempelajari pola tertentu dari citra. Karena itu, setiap channel output biasanya merepresentasikan fitur yang berbeda-beda.

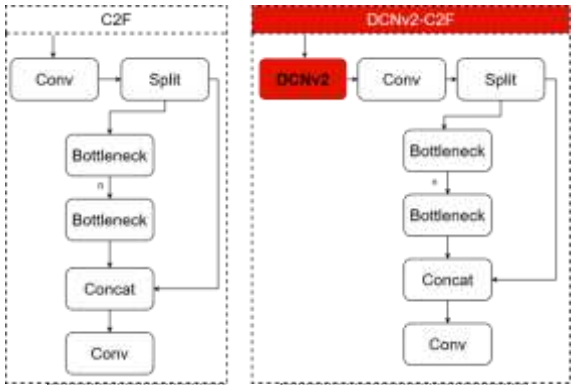
Sebagai contoh:
a) Channel pertama dapat mendeteksi tepi horizontal
b) Channel kedua mendeteksi tepi vertikal
c) Channel lain mendeteksi tekstur cangkang
d) Beberapa channel mendeteksi pola retakan
e) Channel tertentu mendeteksi perbedaan intensitas cahaya hasil *candling*

Semakin banyak jumlah channel, maka semakin banyak variasi fitur yang dapat dipelajari oleh model. Hal ini membuat YOLOv8 mampu mengenali pola kompleks pada objek telur secara lebih akurat. Setelah proses Conv2D selesai dilakukan, output feature map kemudian melewati tahap Batch Normalization untuk menstabilkan distribusi data, lalu diteruskan ke fungsi aktivasi SiLU untuk memberikan sifat non-linear pada jaringan. Gabungan ketiga proses tersebut:
a) Conv2D
b) Batch Normalization
c) Aktivasi SiLU

Menghasilkan feature map yang lebih informatif dan stabil. Feature map tersebut merepresentasikan hasil ekstraksi fitur dari citra input, di mana setiap channel menyimpan informasi pola tertentu yang nantinya akan digunakan model YOLOv8 pada proses deteksi dan klasifikasi objek. Dalam penelitian ini, feature map membantu model mengenali karakteristik retakan telur, tekstur cangkang, bentuk objek, serta pola cahaya hasil pencahayaan *candling* sehingga proses klasifikasi telur retak dan tidak retak dapat dilakukan dengan lebih akurat.

**d. Modul DCN-C2f (*Integrasi Deformable Convolution*)**

*Deformable Convolutional Networks v2* (DCNv2) yang digunakan sebagai modifikasi pada arsitektur YOLO. Pada gambar ditunjukkan perbedaan antara blok C2f standar dan DCNv2-C2f. Perbedaan utamanya terletak pada penambahan layer DeformConv2d pada bagian awal blok sebelum proses *Conv, BatchNorm,* dan *aktivasi SiLU* dilakukan. Pada arsitektur C2f standar, proses ekstraksi fitur masih menggunakan konvolusi biasa dengan posisi kernel tetap (rigid kernel). Kernel hanya mengambil area citra berdasarkan pola grid yang teratur sehingga kurang fleksibel dalam mengikuti bentuk objek yang tidak beraturan. Kondisi ini menjadi keterbatasan ketika mendeteksi retakan telur karena pola retakan memiliki bentuk tipis, melengkung, bercabang, dan tidak selalu berada pada posisi yang sama. Untuk mengatasi masalah tersebut, penelitian ini menggunakan DCNv2. Berbeda dengan Conv2D biasa, DCNv2 mampu mengubah posisi titik sampling kernel secara dinamis mengikuti bentuk objek. Dengan kata lain, kernel tidak lagi mengambil piksel pada posisi tetap, tetapi dapat bergeser menuju area fitur yang dianggap lebih penting. Kemampuan ini membuat proses ekstraksi fitur menjadi lebih adaptif terhadap deformasi objek dan pola retakan yang kompleks.




**Gambar 23. Modul DCN-C2f (*Integrasi Deformable Convolution*)**

**DCNv2:** Modul C2f-DCNv2 merupakan pengembangan dari blok C2f pada arsitektur YOLOv8 dengan menambahkan *Deformable Convolution v2* (DCNv2) pada tahap awal pemrosesan fitur. Penambahan DCNv2 bertujuan untuk meningkatkan kemampuan model dalam mengekstraksi fitur pada objek yang memiliki bentuk tidak beraturan, seperti retakan halus pada permukaan cangkang telur. Pada rancangan ini, feature map masukan terlebih dahulu diproses oleh DCNv2 sebelum diteruskan ke blok C2f, sehingga fitur dapat disesuaikan secara lebih adaptif terhadap perubahan bentuk, posisi, dan pola spasial objek.

Berbeda dengan konvolusi standar yang menggunakan posisi kernel tetap, DCNv2 memiliki mekanisme *offset terdeformasi* $(\Delta p_k)$dan modulasi $(\Delta m_k)$yang dihitung secara dinamis. Mekanisme ini memungkinkan titik sampling kernel bergeser mengikuti kontur dan struktur objek pada citra. Dengan kemampuan tersebut, kernel tidak hanya mengambil informasi pada posisi grid yang tetap, tetapi juga dapat menyesuaikan area sampling pada bagian fitur yang dianggap lebih penting. Dalam penelitian ini, kemampuan adaptif tersebut sangat dibutuhkan karena pola retakan telur umumnya tipis, kecil, melengkung, dan memiliki arah yang berbeda-beda. Proses deformasi kernel pada DCNv2 mengikuti persamaan (1), sehingga proses ekstraksi fitur menjadi lebih fleksibel dibandingkan konvolusi biasa.

Hasil dari proses DCNv2 berupa *feature map* yang lebih adaptif dan kaya informasi spasial. Feature map tersebut kemudian diteruskan ke blok C2f untuk diproses lebih lanjut melalui tahapan *split, Bottleneck, concat,* dan *convolution* akhir. Kombinasi DCNv2 dan C2f memungkinkan model menangkap detail fitur yang lebih kompleks tanpa menghilangkan efisiensi komputasi yang menjadi keunggulan utama YOLOv8. Secara keseluruhan, aliran proses mulai dari input, pemrosesan deformasi pada DCNv2, hingga *feature map* masuk ke blok C2f divisualisasikan pada Gambar 23 sebagai tahapan kerja modul C2f-DCNv:


**Gambar 24. Tahapan DCNv2**

Berikut ini penjelasan mengenai tahapan DCNv2 yang ditampilkan pada Gambar diatas:

**Conv2d:** Lapisan Conv2d pada tahap ini berfungsi untuk menghasilkan parameter *offset* dan *mask* yang dibutuhkan oleh mekanisme *deformable convolution*. *Offset* digunakan untuk menentukan pergeseran posisi titik *sampling kernel*, sedangkan

*mask* berperan dalam mengatur bobot kontribusi setiap titik. Conv2d ini merupakan bagian internal dari DCNv2 dan bukan sekadar lapisan konvolusi biasa. *Offset* ini merepresentasikan pergeseran posisi titik *sampling* dari *grid* konvolusi standar:

$$\Delta P = \text{Conv}(X) \tag{12}$$

*Mask* dihitung melalui operasi konvolusi yang diikuti oleh fungsi aktivasi *sigmoid*, yang dinyatakan sebagai:

$$M = \sigma\big(\text{Conv}(X)\big) \tag{13}$$

Dengan:
$\Delta P$= Pergeseran posisi *kernel*
$M$= *Mask* (0–1)
$\sigma$= Fungsi *sigmoid*

**DeformConv2d:** Tahap awal dimulai dari keluaran modul C2f yang telah menggabungkan dan mengekstraksi fitur dari *layer* sebelumnya. Pada tahap ini, *feature* map memiliki representasi semantik yang kaya, namun masih menggunakan pola *sampling* konvolusi reguler. Integrasi dengan DCNv2 bertujuan untuk meningkatkan fleksibilitas spasial fitur agar lebih adaptif terhadap variasi bentuk dan posisi objek.

1. Nilai Piksel Input

Pada proses *Deformable Convolution v2* (DCNv2), citra input terlebih dahulu direpresentasikan dalam bentuk matriks nilai piksel. Matriks ini merupakan kumpulan intensitas cahaya dari citra digital yang akan diproses untuk mengekstraksi fitur penting pada objek. Misalkan digunakan citra input berukuran $5 \times 5$sebagai berikut:

$$\begin{bmatrix} 12 & 45 & 67 & 23 & 89 \\ 54 & 31 & 78 & 16 & 42 \\ 90 & 25 & 8 & 64 & 37 \\ 8 & 72 & 19 & 95 & 6 \\ 33 & 81 & 14 & 58 & 27 \end{bmatrix}$$

Pada penelitian ini, contoh perhitungan difokuskan pada penggunaan kernel konvolusi berukuran $3 \times 3$untuk mempermudah ilustrasi proses deformable convolution. Pemilihan titik pusat $p_0$pada nilai: Pada proses konvolusi, kernel selalu memiliki satu titik pusat yang dijadikan acuan utama saat proses sampling dilakukan. Pada perhitungan ini dipilih nilai piksel:

$$p_0 = 31$$

Nilai tersebut berada pada koordinat:

$$(1, 1)$$

Koordinat dihitung berdasarkan indeks matriks:

$$\begin{bmatrix} (0,0) & (0,1) & (0,2) & (0,3) & (0,4) \\ (1,0) & (1,1) & (1,2) & (1,3) & (1,4) \\ (2,0) & (2,1) & (2,2) & (2,3) & (2,4) \\ (3,0) & (3,1) & (3,2) & (3,3) & (3,4) \\ (4,0) & (4,1) & (4,2) & (4,3) & (4,4) \end{bmatrix}$$

Sehingga:

$$31 \rightarrow (1,1)$$

Titik pusat ini menjadi posisi awal kernel sebelum dilakukan deformasi (*offset shifting*).

2. Kernel Konvolusi 3×3

Untuk mempermudah ilustrasi, digunakan kernel sederhana berukuran $3 \times 3$dengan seluruh bobot bernilai 1.

Kernel:

$$w = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Kernel ini memiliki:

$$3 \times 3 = 9 \text{ titik sampling.}$$

Pada Conv2D biasa, posisi sampling kernel bersifat tetap (*fixed sampling*). Artinya, setiap titik kernel selalu membaca posisi piksel yang sama terhadap titik pusat.

Sebagai contoh:

$$(-1,-1), (-1,0), (-1,1)$$
$$(0,-1), (0,0), (0,1)$$
$$(1,-1), (1,0), (1,1)$$

Posisi tersebut membentuk area sampling standar di sekitar titik pusat.

Perbedaan Conv2D dan DCNv2 Pada Conv2D biasa:

a) posisi kernel tetap
b) area sampling kaku
c) sulit mengikuti bentuk objek tidak beraturan
d) Sedangkan pada DCNv2, posisi sampling dapat bergeser secara fleksibel menggunakan *offset deformasi.*

3. Offset Deformasi $(\Delta p)$

Pada DCNv2, setiap titik kernel memiliki dua parameter offset:

a) offset horizontal $(\Delta x)$
b) offset vertikal $(\Delta y)$

Offset digunakan untuk menggeser posisi sampling dari posisi normalnya.

Rumus offset:

$$\Delta p_k = (\Delta x_k, \Delta y_k) \quad (14)$$

Keterangan:

a) $\Delta p_k$= offset titik ke-$k$
b) $\Delta x_k$= pergeseran horizontal
c) $\Delta y_k$= pergeseran vertikal

Karena kernel memiliki 9 titik sampling, maka jumlah parameter offset adalah:

$$9 \times 2 = 18$$

Artinya terdapat:

a) 9 offset horizontal
b) 9 offset vertikal

Yang dipelajari secara otomatis oleh jaringan CNN Contoh offset pada kernel:

| Titik Kernel | $\Delta x$ | $\Delta y$ |
|---|---|---|
| (-1,-1) | 0.2 | -0.1 |
| (-1,0) | 0.4 | -0.3 |
| (-1,1) | -0.1 | 0.3 |
| (0,-1) | 0.3 | 0.2 |
| (0,0) | 0.0 | 0.0 |
| (0,1) | -0.2 | 0.4 |
| (1,-1) | 0.1 | -0.2 |
| (1,0) | 0.2 | 0.9 |
| (1,1) | -0.2 | 0.5 |

Interpretasi Offset Offset menunjukkan arah dan jarak pergeseran titik sampling. Sebagai contoh:
Titik kernel $(-1,-1)$
Offset:

$$(0.2,-0.1)$$

Artinya:
a) bergeser 0.2 ke kanan
b) bergeser 0.1 ke atas

Sehingga posisi sampling baru menjadi:

$$(-1+0.2,\ -1-0.1)$$
$$=(-0.8,-1.1)$$

Posisi Sampling Baru Setelah offset ditambahkan, posisi kernel tidak lagi berada tepat pada grid piksel asli. Sebagai contoh:
a) posisi awal: $(1, 0)$
b) setelah offset: $(1.2, 0.9)$

Koordinat tersebut berupa bilangan pecahan (*fractional coordinate*). Masalahnya, citra digital hanya memiliki piksel pada koordinat integer seperti:

$$(1,0),(1,1),(2,0)$$

Dan tidak memiliki piksel langsung pada posisi:

$$(1.2, 0.9)$$

Karena itu diperlukan proses tambahan untuk memperkirakan nilai piksel pada koordinat pecahan
Bilinear Interpolation Setelah posisi sampling baru diperoleh dari proses deformasi kernel, langkah berikutnya adalah menghitung nilai piksel pada posisi tersebut menggunakan bilinear interpolation. Tahapan ini diperlukan karena hasil offset pada DCNv2 sering menghasilkan koordinat pecahan (*non-grid coordinate*), sehingga nilai piksel tidak dapat diambil secara langsung dari matriks feature map. Bilinear interpolation bekerja dengan cara:
a) mencari empat piksel terdekat
b) menghitung kontribusi masing-masing piksel
c) melakukan interpolasi berdasarkan jarak koordinat
d) Dengan metode ini, DCNv2 dapat memperoleh nilai piksel baru secara halus meskipun posisi sampling berada di antara grid piksel asli.

Tahapan ini diperlukan karena hasil offset pada DCNv2 sering menghasilkan koordinat pecahan (non-grid), sehingga nilai piksel tidak dapat diambil secara langsung dari matriks feature map.
4. Rumus DCNv2

Proses deformable convolution mengikuti persamaan:

$$y(p_0)=\sum_{n=1}^{N} w_n \cdot x(p_0+p_n+\Delta p_n)\cdot \Delta m_n \qquad (15)$$

Pada perhitungan ini digunakan:
titik kernel:

$$(x_0,y_0)=(-1,-1)$$

Offset dari tabel:

$$\Delta x = 0.2$$
$$\Delta y = -0.1$$

5. Menghitung Posisi Sampling Baru

Titik pusat feature map:

$$p_0 = (2,2)$$

Maka posisi sampling baru:

$$(x', y') = (x_0 + \Delta x,\ y_0 + \Delta y)$$

Substitusi:

$$x = 2 + (-1) + 0.2$$
$$x = 1.2$$
$$y = 2 + (-1) - 0.1$$
$$y = 0.9$$

Sehingga titik sampling hasil deformasi adalah:

$$(1.2,\ 0.9)$$

Karena koordinat berupa pecahan, maka digunakan bilinear interpolation untuk memperkirakan nilai piksel pada titik tersebut.

6. Menentukan Titik Tetangga

Dari posisi:

$$(1.2,\ 0.9)$$

Diperoleh:

$$x_1 = \lfloor 1.2 \rfloor = 1$$
$$y_1 = \lfloor 0.9 \rfloor = 0$$

Sehingga empat titik tetangga yang digunakan adalah:

$$x(1,0) = 54$$
$$x(2,0) = 90$$
$$x(1,1) = 31$$
$$x(2,1) = 25$$

7. Menghitung Nilai α dan β

Nilai interpolasi horizontal:

$$\alpha = x - x_1$$
$$\alpha = 1.2 - 1$$
$$\alpha = 0.2$$

Nilai interpolasi vertikal:

$$\beta = y - y_1$$
$$\beta = 0.9 - 0$$
$$\beta = 0.9$$

8. Rumus Bilinear Interpolation

Rumus interpolasi:

$$x(1.2,0.9) = (1-\alpha)(1-\beta)x(1,0)$$
$$+\alpha(1-\beta)x(2,0)$$
$$+(1-\alpha)\beta x(1,1)$$
$$+\alpha\beta x(2,1)$$

9. Substitusi Nilai

$$= (0.8)(0.1)(54)$$
$$+(0.2)(0.1)(90)$$
$$+(0.8)(0.9)(31)$$
$$+(0.2)(0.9)(25$$

10. Hasil Perhitungan

Perhitungan tiap komponen:

$$(0.8)(0.1)(54) = 4.32$$
$$(0.2)(0.1)(90) = 1.8$$

$$(0.8)(0.9)(31) = 22.32$$
$$(0.2)(0.9)(25) = 4.5$$

Total:

$$4.32 + 1.8 + 22.32 + 4.5$$
$$= 32.94$$

Maka nilai hasil sampling deformasi adalah:

$$x(1.2,0.9) = 32.94$$

Hasil 32.94 menunjukkan nilai feature map baru yang diperoleh setelah kernel bergeser mengikuti pola objek. Berbeda dengan Conv2D biasa yang hanya mengambil nilai pada posisi grid tetap, DCNv2 mampu mengambil informasi pada posisi non-grid menggunakan bilinear interpolation.

Setelah proses deformasi posisi sampling dan bilinear interpolation dilakukan, tahapan berikutnya pada DCNv2 adalah menghitung mask modulation $(\Delta m)$. Mask ini berfungsi sebagai pengatur kontribusi setiap titik sampling terhadap hasil akhir feature map. Dengan adanya mask, DCNv2 tidak hanya menentukan posisi sampling yang fleksibel, tetapi juga mengatur seberapa besar pengaruh fitur yang diambil pada posisi tersebut.

1. Conv Mask

Pada contoh ini digunakan:

bobot mask:

$$w_m = \begin{bmatrix} 0.01 & 0.01 & 0.01 \\ 0.01 & 0.01 & 0.01 \\ 0.01 & 0.01 & 0.01 \end{bmatrix}$$

bias:

$$bias = 0$$

Patch feature map yang digunakan:

$$\begin{bmatrix} 31 & 78 & 16 \\ 25 & 8 & 64 \\ 72 & 19 & 95 \end{bmatrix}$$

2. Menjumlahkan Nilai Piksel

Jumlah seluruh piksel pada patch:

$$31 + 78 + 16 + 25 + 8 + 64 + 72 + 19 + 95$$
$$= 408$$

3. Menghitung Nilai Conv Mask

Karena semua bobot bernilai 0.01, maka:

$$z_m = 0.01 \times 408$$
$$z_m = 4.08$$

Nilai $z_m$ini menjadi input fungsi sigmoid untuk menghasilkan nilai mask modulation.

4. Menghitung Nilai Mask $(\Delta m)$

Rumus sigmoid:

$$m = \sigma(z_m) = \frac{1}{1 + e^{-z_m}} \tag{16}$$

Substitusi:

$$m = \frac{1}{1 + e^{-4.08}}$$
$$m \approx 0.98$$

Sehingga diperoleh nilai modulation mask:

$$\Delta m = 0.98$$

Karena contoh menggunakan nilai yang sama, maka matriks mask menjadi:

$$\begin{bmatrix} 0.98 & 0.98 & 0.98 \\ 0.98 & 0.98 & 0.98 \\ 0.98 & 0.98 & 0.98 \end{bmatrix}$$

5. Fungsi Mask pada DCNv2

Mask $(\Delta m)$berfungsi untuk menentukan seberapa besar kontribusi hasil deformasi terhadap output akhir.

a) Jika nilai mask mendekati 1 → kontribusi fitur diperkuat.
b) Jika nilai mask mendekati 0 → kontribusi fitur dilemahkan.

Pada contoh ini:

$$\Delta m = 0.98$$

Artinya feature hasil deformasi memiliki kontribusi yang sangat besar terhadap output akhir.

6. Menghitung Output Akhir DCNv2

Sebelumnya diperoleh hasil bilinear interpolation:

$$x(1.2,0.9) = 32.94$$

Kemudian dikalikan dengan mask:

$$x_{final} = \Delta m \times x(1.2,0.9)$$
$$x_{final} = 0.98 \times 32.94$$
$$x_{final} = 32.28$$

Hasil akhir sebesar 32.28 menunjukkan nilai feature map setelah proses deformasi dan modulation pada DCNv2. Nilai ini menjadi output yang lebih adaptif karena kernel tidak hanya bergeser mengikuti bentuk objek, tetapi juga menyesuaikan kontribusi fitur penting pada area retakan.

*Batch Normalization*: *Output* dari DeformConv2d selanjutnya diproses menggunakan *Batch Normalization*. Tahap ini bertujuan untuk menormalkan distribusi nilai fitur, sehingga dapat mengurangi masalah internal *covariate shift*. Dengan demikian, proses pelatihan menjadi lebih stabil dan konvergensi model dapat dicapai dengan lebih cepat.

Normalisasi dilakukan dengan persamaan:

$$\hat{Y} = \frac{Y - \mu}{\sqrt{\sigma^2 + \epsilon}} \tag{17}$$

Kemudian diskalakan:

$$Z = \gamma \hat{Y} + \beta \tag{18}$$

Dengan:

$\mu, \sigma^2$= *mean* dan varian

$\gamma, \beta$= parameter skala dan geser

Aktivasi SiLU: Tahap terakhir adalah penerapan fungsi aktivasi SiLU (*Sigmoid Linear Unit*). Fungsi ini memberikan non linearitas pada jaringan dengan cara mengalikan *input* dengan fungsi *sigmoid*. Penggunaan SiLU membantu meningkatkan kemampuan model dalam mempelajari pola yang kompleks serta menjaga aliran *gradien* tetap stabil.Secara matematis, fungsi SiLU dirumuskan sebagai:

$$\text{SiLU}(x) = x \cdot \sigma(x) \tag{19}$$

Dengan:

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{20}$$

Dengan:

$x$: nilai *input* hasil *Batch Normalization*

$\sigma(x)$: fungsi *sigmoid* yang membatasi nilai antara 0 dan 1
Tahap Split:
Setelah feature map diproses menggunakan DCNv2, tahap berikutnya pada blok C2f adalah proses split feature atau pembagian kanal fitur. Pada tahap ini, feature map masukan dibagi menjadi beberapa bagian kanal untuk diproses melalui jalur yang berbeda. Tujuan proses split adalah agar model dapat mempertahankan informasi fitur awal sekaligus melakukan ekstraksi fitur baru secara lebih efisien.
Proses pembagian fitur dinyatakan dengan persamaan:

$$[A, B] = split(C) \tag{21}$$

Keterangan:
$C$= jumlah channel masukan,
$A, B$= hasil pembagian channel,
$split(c)$= fungsi pemisahan kanal fitur.
Pada contoh gambar, feature map hasil sebelumnya digunakan sebagai input:

$$\begin{bmatrix} 31 & 78 & 16 \\ 25 & 8 & 64 \\ 72 & 19 & 95 \end{bmatrix}$$

Kemudian dilakukan proses conv 1×1 untuk menghasilkan feature map pada beberapa channel.
1. Channel 1
Pada channel pertama digunakan:

$$X^{(1)} = 1 \times X$$

Sehingga feature map yang dihasilkan tetap:

$$\begin{bmatrix} 31 & 78 & 16 \\ 25 & 8 & 64 \\ 72 & 19 & 95 \end{bmatrix}$$

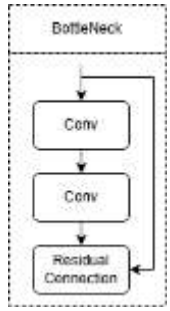
Channel ini mempertahankan informasi fitur asli dari input sebelumnya.
2. Channel 2
Pada channel kedua digunakan:

$$X^{(2)} = 2 \times X$$

Sehingga diperoleh feature map baru:

$$\begin{bmatrix} 2 & 4 & 6 \\ 8 & 10 & 12 \\ 14 & 16 & 18 \end{bmatrix}$$

Channel kedua digunakan untuk menghasilkan representasi fitur tambahan agar model dapat mempelajari pola yang lebih beragam.
*Bottleneck*:




Gambar 25. Tahapan *BottleNeck*

Setelah feature map dibagi melalui proses split, tahap berikutnya adalah Bottleneck. Pada tahap ini, feature map diproses menggunakan beberapa lapisan konvolusi untuk memperdalam ekstraksi fitur tanpa meningkatkan komputasi secara berlebihan. Struktur Bottleneck terdiri atas:

a) Conv 1×1
b) Conv 3×3
c) Residual Connection

Pada gambar, proses Bottleneck mengikuti persamaan:

$$Y = X + f_{3\times3}\left(f_{1\times1}(X)\right) \tag{22}$$

Keterangan:

$X$= feature map masukan,
$f_{1\times1}$= proses konvolusi 1×1,
$f_{3\times3}$= proses konvolusi 3×3,
$Y$= output Bottleneck.

Feature map masukan yang digunakan adalah:

$$X = \begin{bmatrix} 31 & 78 & 16 \\ 25 & 8 & 64 \\ 72 & 19 & 95 \end{bmatrix}$$

1. Conv 1×1

Tahap pertama adalah Conv 1×1 yang bertujuan mengatur dan mengombinasikan informasi antar channel tanpa mengubah ukuran spasial feature map. Misalkan digunakan bobot sederhana:

$$w_{1\times1} = 0.5$$

Maka hasil Conv 1×1:

$$f_{1\times1}(X) = 0.5 \times X = \begin{bmatrix} 15.5 & 39 & 8 \\ 12.5 & 4 & 32 \\ 36 & 9.5 & 47.5 \end{bmatrix}$$

2. Conv 3×3

Output Conv 1×1 kemudian diproses menggunakan Conv 3×3 untuk menangkap pola spasial dan hubungan antar piksel di sekitar feature map. Misalkan kernel yang digunakan:

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Karena ukuran feature map dan kernel sama-sama 3×3, maka seluruh nilai dijumlahkan:

$$15.5 + 39 + 8 + 12.5 + 4 + 32 + 36 + 9.5 + 47.5 = 204$$

Nilai ini menjadi hasil transformasi Conv 3×3:

$$f_{3\times3}(f_{1\times1}(X)) = 204$$

3. Residual Connection

Tahap terakhir adalah Residual Connection, yaitu menambahkan kembali feature map awal $X$dengan hasil transformasi konvolusi. Tujuan residual connection adalah menjaga informasi penting dari input awal agar tidak hilang selama proses ekstraksi fitur. Sesuai persamaan:

$$Y = X + f_{3\times3}\left(f_{1\times1}(X)\right) \tag{23}$$

Maka:

$$Y = X + 204$$

Hasil output Bottleneck:

$$\begin{bmatrix} 235 & 282 & 220 \\ 229 & 212 & 268 \\ 276 & 223 & 299 \end{bmatrix}$$

Concat: Setelah feature map diproses melalui Bottleneck, tahap berikutnya pada blok C2f adalah concat atau penggabungan feature map. Tahap ini bertujuan untuk menggabungkan informasi fitur dari beberapa channel agar representasi fitur menjadi lebih kaya sebelum masuk ke proses konvolusi akhir. terdapat dua feature map hasil proses sebelumnya, yaitu:

Channel 1

$$C_{in1} = \begin{bmatrix} 235 & 282 & 220 \\ 229 & 212 & 268 \\ 276 & 223 & 299 \end{bmatrix}$$

Channel 2

$$C_{in2} = \begin{bmatrix} 2 & 4 & 6 \\ 8 & 10 & 12 \\ 14 & 16 & 18 \end{bmatrix}$$

1. Proses Concat

Penggabungan feature map dilakukan menggunakan operasi:

$$C_{out} = [C_{in1}, C_{in2}] \quad (24)$$

Pada proses concat:

a) ukuran spasial tidak berubah,
b) jumlah channel bertambah.

Sebelum concat setiap feature map memiliki ukuran:

$$3 \times 3 \times 1$$

Karena terdapat dua channel, maka setelah concat:

$$3 \times 3 \times 2$$

Artinya tinggi = 3, lebar = 3, jumlah channel = 2

2. Bentuk Output Concat

Hasil concat dapat direpresentasikan sebagai dua layer feature map yang ditumpuk menjadi satu output:

Channel pertama

$$\begin{bmatrix} 235 & 282 & 220 \\ 229 & 212 & 268 \\ 276 & 223 & 299 \end{bmatrix}$$

Channel kedua

$$\begin{bmatrix} 2 & 4 & 6 \\ 8 & 10 & 12 \\ 14 & 16 & 18 \end{bmatrix}$$

Sehingga output concat:

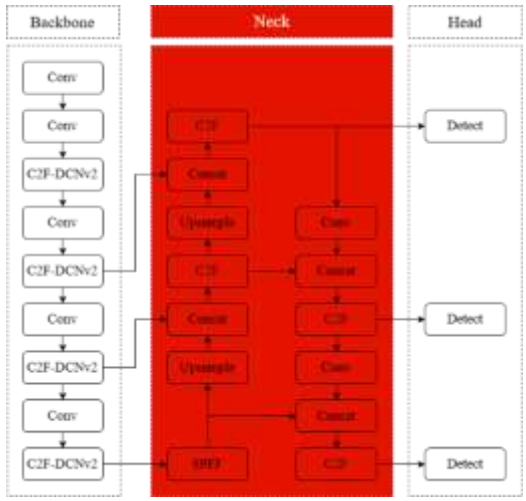
$$C_{out} \in \mathbb{R}^{3\times3\times2}$$

3. Fungsi Concat pada C2f

Tahap concat berfungsi untuk menggabungkan fitur lama dan fitur baru sehingga informasi penting dari proses sebelumnya tetap dipertahankan. Proses ini juga bertujuan memperkaya representasi feature map agar model memiliki kemampuan ekstraksi fitur yang lebih baik. Pada blok C2f, concat memungkinkan model mengombinasikan fitur awal, fitur hasil Bottleneck, serta fitur hasil transformasi lainnya ke dalam satu feature map. Dengan penggabungan tersebut, informasi yang diteruskan ke layer berikutnya menjadi lebih lengkap dan lebih representatif dalam menggambarkan pola objek pada.

### e. Tahapan *Neck*

Tahapan *neck* merupakan bagian penting dalam arsitektur model deteksi objek yang berperan sebagai penghubung antara *backbone* dan *head*. Pada tahap ini, fitur-fitur yang telah diekstraksi oleh *backbone* dengan berbagai tingkat resolusi dan kedalaman diproses lebih lanjut untuk menghasilkan representasi fitur yang lebih kaya dan informatif.



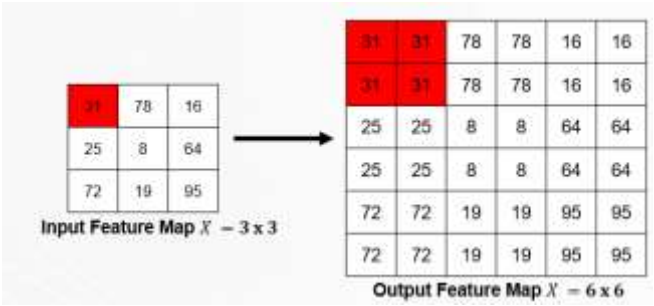

**Gambar 26. Tahapan Neck**

**C2f :** berfungsi sebagai modul pemrosesan lanjutan setelah penggabungan fitur. Modul ini bertugas untuk menyaring, menguatkan, dan menstabilkan fitur hasil *Concat* agar lebih representatif dan efisien. Penggunaan C2f di *neck* membantu mengurangi redundansi fitur sekaligus menjaga aliran gradien selama proses pelatihan, sehingga performa deteksi menjadi lebih optimal.

***Concat* :** Setelah proses *upsample*, *feature map* dengan ukuran yang sama digabungkan menggunakan operasi *Concat*. Penggabungan ini dilakukan pada dimensi *channel* sehingga informasi dari berbagai skala dapat dipadukan dalam satu *feature map*. *Concat* berperan dalam menyatukan fitur detail dan fitur semantik, sehingga konteks objek dapat dipahami dengan lebih baik oleh model.

***Upsampel* :** pada bagian *neck* digunakan untuk meningkatkan resolusi *feature map* dari level yang lebih dalam agar memiliki ukuran spasial yang sama dengan *feature map* dari level yang lebih dangkal. Proses ini memungkinkan terjadinya penggabungan fitur antar level. *Upsample* sangat penting dalam mempertahankan informasi spasial, khususnya untuk meningkatkan kemampuan deteksi objek berukuran kecil. Proses *Upsampling* pada arsitektur YOLOv8 berfungsi untuk memperbesar dimensi spasial (*height* dan *width*) dari peta fitur yang sebelumnya mengalami pengecilan melalui operasi konvolusi atau *downsampling*. Tahap ini menjadi bagian penting dalam mekanisme *feature fusion* pada jalur *neck*, di mana fitur dari beberapa level resolusi digabungkan untuk menghasilkan representasi yang lebih kaya dan informatif. Dalam implementasinya, YOLOv8 memanfaatkan fungsi *torch.nn.Upsample* dari pustaka PyTorch dengan parameter mode= *nearest*, sehingga proses interpolasi dilakukan dengan metode *nearest-neighbor*. Teknik ini memperbesar ukuran peta fitur dengan cara mereplikasi nilai piksel dari posisi terdekat tanpa melakukan perhitungan interpolasi yang lebih kompleks. Setiap elemen pada fitur asal diperluas secara horizontal dan vertikal berdasarkan nilai

*scale factor*, misalnya dua kali lipat ketika *scale_factor*=2. Ilustrasi *Upsamlpling* tersebut bisa dilihat pada Gambar 27 di bawah ini:

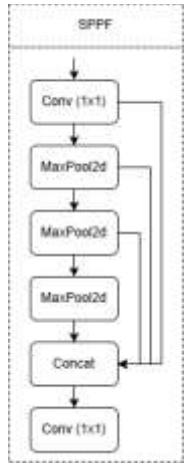



**Gambar 27. Ilustrasi Upsampling**

Peta fitur awal memiliki ukuran 3x3. Setiap piksel dari peta fitur awal direplikasi menjadi blok berukuran 6x6 dengan nilai yang sama, sehingga struktur spasial tetap konsisten meskipun ukuran diperbesar. Contoh visual transformasi ini dapat dilihat pada Gambar 27, yang menunjukkan proses replikasi nilai piksel dari resolusi rendah ke resolusi yang lebih tinggi menggunakan pendekatan *nearest neighbor interpolation*. Proses *upsampling* ini berperan penting dalam meningkatkan efektivitas penggabungan fitur pada YOLOv8, sehingga mendukung deteksi objek yang lebih akurat pada berbagai skala.

**SPFF:** Modul SPPF (*Spatial Pyramid Pooling Fast*) pada arsitektur YOLOv11 berfungsi untuk memperluas konteks spasial fitur yang dihasilkan oleh lapisan sebelumnya tanpa mengubah ukuran resolusi *feature map*. Modul ini dirancang untuk menangkap informasi objek pada berbagai skala secara efisien, khususnya pada objek berukuran kecil hingga besar. Proses pada modul SPPF diawali dengan *feature map* hasil konvolusi. *Feature map* tersebut kemudian diproses melalui tiga tahap *max pooling* secara berurutan. Berbeda dengan *Spatial Pyramid Pooling* konvensional yang menggunakan beberapa kernel berbeda, SPPF menggunakan kernel *pooling* yang sama tetapi diaplikasikan secara bertingkat. Setiap tahap *pooling* mempertahankan ukuran spasial yang sama (karena *stride* = 1 dan *padding* sesuai), namun secara efektif memperluas *receptive field*, sehingga informasi dari area sekitar yang semakin luas dapat diekstraksi.

Setelah melalui tiga tahap *max pooling*, seluruh *feature map* yang dihasilkan termasuk *feature map* awal sebelum *pooling* digabungkan (*concatenate*) pada dimensi kanal. Proses penggabungan ini menghasilkan representasi fitur yang mengandung informasi lokal dan global secara bersamaan. Selanjutnya, hasil *concatenation* diproses menggunakan konvolusi akhir untuk menyesuaikan jumlah kanal keluaran agar kompatibel dengan lapisan berikutnya. Tahapan SPPF tersebut bisa dilihat pada Gambar 28 di bawah ini:




**Gambar 28. Tahapan SPPF**

**Input Feature Map pada SPPF**

Setelah feature map diproses melalui blok Conv dan C2f-DCNv2, tahap berikutnya adalah modul SPPF (Spatial Pyramid Pooling Fast). Modul ini menerima feature map hasil ekstraksi fitur sebelumnya yang telah mengandung informasi spasial dan semantik dari citra input. Pada contoh gambar, feature map masukan memiliki ukuran:

$$X \in \mathbb{R}^{8\times 8}$$

Dengan nilai:

$$\begin{bmatrix} 12 & 45 & 67 & 23 & 89 & 34 & 56 & 78 \\ 54 & 31 & 78 & 16 & 42 & 91 & 25 & 60 \\ 90 & 25 & 8 & 64 & 37 & 48 & 72 & 14 \\ 8 & 72 & 19 & 95 & 6 & 55 & 41 & 83 \\ 33 & 81 & 14 & 58 & 27 & 66 & 92 & 21 \\ 47 & 10 & 39 & 74 & 88 & 5 & 63 & 50 \\ 29 & 84 & 61 & 17 & 46 & 90 & 11 & 68 \\ 70 & 24 & 52 & 36 & 79 & 13 & 94 & 32 \end{bmatrix}$$

Keterangan:

$H$= tinggi feature map,
$W$= lebar feature map,
$C$= jumlah channel.

Secara umum, feature map input dinyatakan sebagai:

$$X \in \mathbb{R}^{H\times W\times C}$$

Pada tahap ini, SPPF tidak berfokus mengubah ukuran feature map, tetapi memperluas receptive field agar model dapat menangkap konteks spasial yang lebih luas di sekitar objek.

Max Pooling Bertahap pada SPPF

Feature map input kemudian diproses menggunakan max pooling bertahap. Setiap pooling dilakukan secara berurutan menggunakan kernel yang sama. Tujuan proses ini adalah mengambil nilai maksimum pada area tertentu sehingga fitur paling dominan tetap dipertahankan.

Tahapan pooling dilakukan sebagai berikut:

$$P_1 = MaxPool(X)$$
$$P_2 = MaxPool(P_1)$$
$$P_3 = MaxPool(P_2)$$

Pooling dilakukan bertahap hingga menghasilkan feature map dengan cakupan informasi yang semakin luas.

**1. Pooling Pertama $(\boldsymbol{P_1})$**

Feature map $X$diproses menggunakan max pooling sehingga menghasilkan:

$$P_1 = \begin{bmatrix} 54 & 78 & 91 & 78 \\ 90 & 95 & 55 & 83 \\ 81 & 74 & 88 & 92 \\ 84 & 61 & 90 & 94 \end{bmatrix}$$

Setiap nilai merupakan nilai maksimum dari area pooling pada feature map sebelumnya.

Contoh:

Pada area:

$$\begin{bmatrix} 12 & 45 \\ 54 & 31 \end{bmatrix}$$

Nilai maksimum adalah:

$$54$$

**2. Pooling Kedua $(\boldsymbol{P_2})$**

Output $P_1$kemudian diproses kembali menggunakan max pooling:

$$P_2 = \begin{bmatrix} 95 & 91 \\ 84 & 94 \end{bmatrix}$$

Contoh:

Pada area:

$$\begin{bmatrix} 54 & 78 \\ 90 & 95 \end{bmatrix}$$

Nilai maksimum:

$$95$$

**3. Pooling Ketiga $(\boldsymbol{P_3})$**

Feature map hasil sebelumnya diproses kembali:

$$P_3 = [95]$$

Nilai tersebut menjadi representasi fitur paling dominan dari keseluruhan feature map.

***Concat*:** Setelah melewati seluruh tahap *max pooling*, *feature map* awal dan hasil dari setiap tahap *pooling* digabungkan pada dimensi kanal. Proses *concatenation* ini bertujuan untuk mengintegrasikan informasi lokal dan global dalam satu representasi fitur. Dengan cara ini, jaringan memperoleh fitur multi skala yang lebih kaya dan representatif.

Seluruh *feature map* digabungkan pada dimensi kanal:

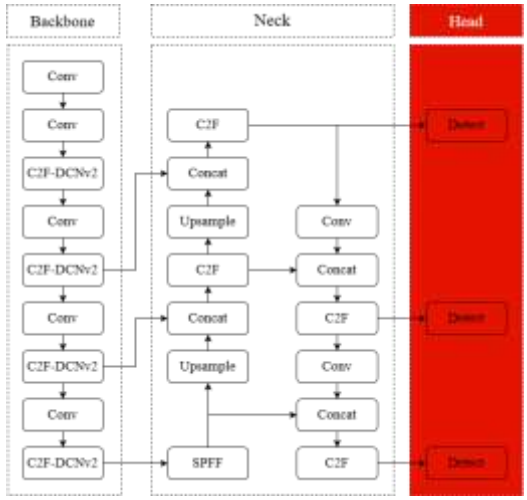
$$C_{\text{cat}} = \text{Concat}(X, P_1, P_2, P_3) \quad (25)$$

***Convolusi* Akhir:** Hasil penggabungan fitur selanjutnya diproses menggunakan lapisan konvolusi akhir. Konvolusi ini berfungsi untuk menyesuaikan jumlah kanal keluaran serta menyatukan informasi hasil *concatenation* menjadi *feature map* yang lebih kompak dan siap diteruskan ke lapisan berikutnya pada arsitektur jaringan. Hasil *concatenation* kemudian diproses dengan konvolusi untuk menyesuaikan kanal:

$$Y = \text{Conv}(C_{\text{cat}}) \quad (26)$$

### f. Tahapan *Head*

Tahapan *head* merupakan bagian akhir dalam arsitektur model deteksi objek yang berfungsi menghasilkan prediksi akhir berdasarkan fitur yang telah diproses pada tahap *neck*. Pada tahap ini, representasi fitur multi-skala digunakan untuk melakukan tugas utama deteksi, yaitu menentukan lokasi objek melalui *bounding box*, mengklasifikasikan jenis objek, serta menghitung tingkat kepercayaan (*confidence score*) dari setiap prediksi. Tahapan *Head* bisa dilihat pada Gambar 29 di bawah ini:




**Gambar 29. Tahapan *Head***

***Head mode Training*:** proses pelatihan model deteksi objek, bagian *Head* memiliki peran penting dalam mengonversi fitur hasil ekstraksi menjadi prediksi yang dapat dibandingkan dengan data kebenaran (*ground truth*). Pada mode *training*, *Head* tidak hanya berfungsi menghasilkan prediksi, tetapi juga menjadi komponen utama dalam proses pembelajaran melalui perhitungan *loss* dan pembaruan bobot jaringan. Oleh karena itu, pemahaman terhadap alur kerja *Head* pada mode *training* perlu diawali dengan penjelasan mengenai tahap awal, yaitu proses pemberian data *input* ke dalam model.

**1. *Input* Data** Pada bagian *input* data memasukkan data berupa citra *input* beserta label (*ground truth*) ke dalam model. Citra berfungsi sebagai sumber informasi visual, sedangkan label menyediakan informasi kebenaran yang menjadi acuan pembelajaran model. Label tersebut mencakup kelas objek yang menunjukkan kategori objek pada citra serta koordinat *bounding box* yang merepresentasikan posisi dan ukuran objek secara spasial. Pada tahap ini, data masih berada dalam bentuk citra mentah dan anotasi asli, sehingga belum mengalami proses ekstraksi atau transformasi fitur. Informasi ini kemudian digunakan oleh model untuk mempelajari keterkaitan antara pola visual pada citra dengan lokasi serta kategori objek yang benar selama proses pelatihan.

**2. *Forward Pass - Backbone*** Setelah data citra dan label dimasukkan ke dalam model, citra *input* selanjutnya diproses oleh bagian *backbone* melalui serangkaian lapisan konvolusi. *Backbone* berfungsi untuk mengekstraksi fitur visual dasar yang terkandung dalam citra, seperti tepi (*edge*), tekstur, dan pola visual lainnya. Proses ini memungkinkan model menangkap karakteristik penting dari objek yang terdapat pada citra. Hasil pemrosesan *backbone* berupa *feature map* bertingkat yang

merepresentasikan informasi visual pada berbagai tingkat kedalaman, mulai dari fitur sederhana hingga fitur yang lebih kompleks, yang kemudian diteruskan ke bagian *neck* untuk proses fusi dan penguatan fitur.

**3. *Forward Pass - Neck*** *Feature map* yang dihasilkan oleh *backbone* selanjutnya diteruskan ke bagian *neck*. *Neck* berperan dalam menggabungkan fitur dari berbagai tingkat resolusi untuk memperkuat representasi visual objek pada citra. Melalui proses fusi *multi skala* (*multi scale fusion*), neck mampu mengombinasikan informasi detail dari *feature map* beresolusi tinggi dengan informasi kontekstual dari *feature map* beresolusi lebih rendah. Proses ini melibatkan beberapa operasi utama, seperti *upsample* untuk menyamakan ukuran spasial *feature map*, *concat* untuk menggabungkan fitur dari jalur berbeda, modul C2F untuk meningkatkan efisiensi dan kedalaman representasi fitur, serta modul SPPF untuk memperluas konteks spasial tanpa mengubah resolusi. Hasil akhir dari proses pada *neck* adalah *feature map* multi skala yang telah diperkaya informasi dan siap digunakan oleh bagian *head* untuk melakukan prediksi deteksi objek.

**4. *Forward Pass - Head*** *Feature map* multi skala yang dihasilkan oleh *neck* selanjutnya diteruskan ke bagian *head*. Pada tahap ini, *feature map* diproses melalui dua cabang utama, yaitu cabang regresi (cv2) dan cabang klasifikasi (cv3). Cabang regresi bertugas memprediksi parameter koordinat *bounding box* yang mencakup posisi pusat serta ukuran objek, sedangkan cabang klasifikasi berfungsi untuk memprediksi kelas objek yang terdeteksi. Hasil keluaran dari kedua cabang tersebut kemudian digabungkan menggunakan operasi *concatenation* sehingga membentuk prediksi mentah (*raw predictions*). Prediksi mentah ini merepresentasikan informasi deteksi awal yang akan digunakan pada tahap selanjutnya dalam proses *training* maupun *inference*.

**5. Mode *Training* Aktif** Pada saat model berada dalam mode *training*, hasil keluaran dari *head* masih berupa prediksi mentah. Prediksi ini tidak langsung diubah menjadi *bounding box* sebenarnya pada citra, melainkan digunakan terlebih dahulu untuk menghitung nilai *loss*. Nilai *loss* ini menunjukkan seberapa besar perbedaan antara hasil prediksi model dengan data kebenaran (*ground truth*). Proses inilah yang membedakan mode *training* dengan mode *inference*, di mana pada *inference* prediksi akan langsung didecoding dan ditampilkan sebagai hasil deteksi.

**6. Perhitungan *Loss*** Pada tahap selanjutnya, prediksi mentah yang dihasilkan oleh model dibandingkan dengan data kebenaran (*ground truth*) untuk menghitung nilai *loss*, *Loss* Klasifikasi,*Loss* Tambahan.

**7. Loss Klasifikasi** Digunakan untuk mengukur seberapa tepat model dalam memprediksi kelas objek yang terdeteksi dibandingkan dengan label kelas pada *ground truth*. Pada tahap ini, probabilitas kelas hasil prediksi dibandingkan dengan kelas yang sebenarnya untuk mengevaluasi tingkat kesalahan klasifikasi. Umumnya, *loss* klasifikasi dihitung menggunakan *Cross-Entropy Loss*, karena efektif dalam mengukur perbedaan antara distribusi probabilitas prediksi dan distribusi kelas yang benar. Untuk kasus klasifikasi multi kelas, *Cross-Entropy Loss* dirumuskan sebagai:

$$\mathcal{L}_{\text{cls}} = -\sum_{i=1}^{N} y_i \log(p_i) \tag{27}$$

Dimana:

$N$: Jumlah kelas objek

$y_i$: Label *ground truth* untuk kelas ke-$i$ (bernilai 1 untuk kelas benar, 0 untuk kelas lain)

$p_i$: Probabilitas prediksi model untuk kelas ke-$i$

Semakin besar probabilitas yang diberikan model pada kelas yang benar, semakin kecil nilai *loss* klasifikasi yang dihasilkan. Sebaliknya, jika model memberikan probabilitas rendah pada kelas yang benar, nilai *loss* akan meningkat. Dengan demikian, loss klasifikasi mendorong model untuk menghasilkan prediksi kelas yang akurat dan konsisten selama proses *training*.

Dalam kasus deteksi telur ayam, objek yang dianalisis dibagi menjadi dua kelas utama, yaitu Telur Utuh dan Telur Retak. Klasifikasi ini bertujuan untuk membedakan kondisi fisik telur, sehingga model dapat mengenali apakah telur masih dalam keadaan baik atau mengalami keretakan *Ground Truth* Objek yang ada: Telur Retak

$$y = [0,1]$$

Nilai 0 pada indeks pertama menunjukkan bahwa objek bukan Telur Utuh. Nilai 1 pada indeks kedua menunjukkan bahwa objek benar adalah Telur Retak.
Prediksi Model (probabilitas *softmax*)

$$p = [0.3,0.7]$$

Probabilitas 0.3 pada kelas Telur Utuh menunjukkan keyakinan model rendah bahwa telur termasuk kelas utuh. Probabilitas 0.7 pada kelas Telur Retak menunjukkan model cukup yakin bahwa objek adalah Telur Retak.
*Bounding Box*

*Ground truth*: $B_{gt} = (x_{\min} = 2, y_{\min} = 2, x_{\max} = 6, y_{\max} = 6)$
Prediksi: $B_{pred} = (x_{\min} = 3, y_{\min} = 3, x_{\max} = 7, y_{\max} = 6)$

Hitung *Cross-Entropy Loss* (Klasifikasi) menggunakan persamaan 30:
Substitusi nilai:

$$\mathcal{L}_{\text{cls}} = -(0 \cdot \log\ 0.3 + 1 \cdot \log\ 0.7) = -\log\ 0.7$$

Hitung numerik:

$$\log 0.7 \approx -0.3567$$
$$\mathcal{L}_{\text{cls}} = -(-0.3567) = 0.3567$$

Kesimpulan: *Loss* klasifikasi = 0.357, menunjukkan model cukup yakin bahwa telur retak, tetapi belum sempurna.

**8. *Loss* Regresi Bounding Box**

*Loss* regresi *bounding box* digunakan untuk mengukur seberapa tepat model dalam memprediksi posisi dan ukuran *bounding box* dibandingkan dengan *ground truth*, yang mencakup kesesuaian koordinat pusat objek serta dimensi lebar dan tinggi *bounding box*. Salah satu pendekatan yang umum digunakan dalam perhitungan *loss* regresi adalah *Intersection over Union* (IoU), yaitu ukuran tingkat tumpang tindih antara *bounding box* hasil prediksi ($B_{pred}$) dan *bounding box ground truth* ($B_{gt}$). Secara matematis, IoU menggunakan persamaan 5

Nilai IoU berada pada rentang $0 \leq \text{IoU} \leq 1$, di mana nilai yang lebih besar menunjukkan tingkat kesesuaian posisi dan ukuran *bounding box* yang lebih baik. Berdasarkan nilai IoU tersebut, *loss regresi bounding box* umumnya dirumuskan sebagai fungsi kebalikan dari IoU, yaitu:

$$\mathcal{L}_{\text{bbox}} = 1 - \text{IoU}$$

Dengan formulasi ini, semakin kecil nilai *loss regresi bounding box*, semakin besar nilai IoU yang diperoleh, yang menandakan bahwa prediksi *bounding box* semakin mendekati *ground truth*. Pendekatan berbasis IoU ini lebih efektif dibandingkan regresi koordinat langsung karena secara langsung mengukur kesesuaian spasial antara *bounding box* prediksi dan objek sebenarnya pada

citra.berikut ini contoh perhitungan IoU Misalkan kita memiliki citra yang berisi satu telur ayam, dengan *ground truth bounding box* dan prediksi model sebagai berikut:

*Ground truth* (B_gt):
Misalkan *bounding box* yang digunakan untuk menandai telur di citra:
*Ground truth*: $B_{gt} = (x_{\min} = 2, y_{\min} = 2, x_{\max} = 6, y_{\max} = 6)$

Prediksi: $B_{pred} = (x_{\min} = 3, y_{\min} = 3, x_{\max} = 7, y_{\max} = 6)$
Hitung *Intersection*:

$$x_{\min} = \max(2,3) = 3, y_{\min} = \max(2,3) = 3$$
$$x_{\max} = \min(6,7) = 6, y_{\max} = \min(6,6) = 6$$
$$\text{Area}_{\text{inter}} = (6-3) \cdot (6-3) = 9$$

Hitung *Union*:

$$\textit{Area}_{\textit{pred}} = (7-3) \cdot (6-3) = 12$$
$$\textit{Area}_{\textit{gt}} = (6-2) \cdot (6-2) = 16$$
$$\textit{Area}_{\textit{union}} = 12 + 16 - 9 = 19$$

IoU dan *Loss Bounding Box*:

$$\textit{IoU} = \frac{9}{19} \approx 0.474$$
$$\mathcal{L}_{\textit{bbox}} = 1 - 0.474 \approx 0.526$$

Hitung *loss bounding box*

$$\mathcal{L}_{\text{bbox}} = 1 - \text{IoU} = 1 - 0.526 \approx 0.474$$

Berdasarkan perhitungan, prediksi *bounding box* untuk kelas Telur Retak memiliki nilai IoU sebesar 0,474, yang menunjukkan bahwa posisi dan ukuran *bounding box* model masih sedikit bergeser dibandingkan *ground truth*. Nilai *loss bounding box* sebesar 0,526 menandakan bahwa model perlu melakukan perbaikan untuk meningkatkan akurasi prediksi posisi dan ukuran objek. Dengan demikian, meskipun model berhasil mendeteksi keberadaan telur retak, prediksi *bounding box* belum optimal dan perlu disempurnakan agar kesesuaian dengan *ground truth* lebih tinggi.

**9. *Loss* Tambahan: *Distribution Focal Loss* (DFL)** merupakan *loss* tambahan yang digunakan untuk meningkatkan ketelitian prediksi bounding box, khususnya dalam merepresentasikan koordinat posisi secara lebih presisi. Berbeda dengan regresi *bounding box* konvensional yang memprediksi nilai koordinat secara langsung, DFL memodelkan setiap koordinat *bounding box* sebagai distribusi probabilitas diskret. Pendekatan ini memungkinkan model mempelajari ketidakpastian posisi dan menghasilkan prediksi yang lebih halus. Secara matematis, jika suatu koordinat *bounding box* di representasikan sebagai distribusi probabilitas $p_i$pada $n$bin diskret dengan nilai target $y$, maka *Distribution Focal Loss* dirumuskan sebagai:

$$\mathcal{L}_{\text{DFL}} = -\sum_{i=1}^{n} [y_i \log(p_i)] \quad (28)$$

Di mana:
$p_i$: Probabilitas prediksi pada bin ke-$i$
$y_i$: Distribusi target yang dibentuk dari nilai *ground truth*

Melalui DFL, model tidak hanya mempelajari lokasi *bounding box* secara kasar, tetapi juga mampu memperkirakan posisi objek dengan resolusi yang lebih tinggi.

Oleh karena itu, penggunaan DFL dapat meningkatkan akurasi regresi *bounding box*, terutama pada objek kecil atau kasus dengan perbedaan posisi yang sangat halus. Berikut ini contoh perhitungan DFL Misal koordinat x di prediksi menggunakan 5 bin:
Target distribusi (*ground truth*):

$$y = [0,0.7,0.3,0,0]$$

Prediksi model:

$$p = [0.1,0.6,0.2,0,05,0,05]$$

$$\mathcal{L}_{\text{DFL}} = -(0 \cdot \log 0.1 + 0.7 \cdot \log 0.6 + 0.3 \cdot \log 0.2) \approx 0.840$$

Interpretasi: Prediksi koordinat x relatif mendekati distribusi target, tetapi masih ada kesalahan posisi.
Total *Loss*

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{cls}} + \mathcal{L}_{\text{bbox}} + \mathcal{L}_{\text{DFL}} \approx 0.357 + 0.526 + 0.840 = 1.723$$

Total *loss* sebesar 1.723 menunjukkan bahwa model perlu memperbaiki prediksi posisi *bounding box* sambil mempertahankan keyakinan pada kelas yang benar.

**10. *Backpropagation***

Setelah nilai *loss* diperoleh, proses selanjutnya adalah *backpropagation*. Pada tahap ini, nilai *loss* digunakan untuk menghitung gradien terhadap setiap bobot dalam jaringan menggunakan algoritma *backpropagation*. Gradien yang dihasilkan menunjukkan arah perubahan bobot serta besar penyesuaian yang diperlukan agar nilai *loss* dapat di minimalkan. Dengan adanya informasi gradien ini, model dapat mengetahui bobot mana yang perlu diperbesar atau diperkecil sehingga prediksi yang dihasilkan pada iterasi berikutnya menjadi lebih akurat.

Data Awal (Penentuan Parameter Awal) Sebelum proses pelatihan dimulai, model harus memiliki nilai awal. Nilai ini biasanya diberikan secara acak atau ditentukan diawal pelatihan. Nilai awal tersebut digunakan sebagai titik awal untuk menghitung prediksi dan *error*.
Rumus dan Nilai

$$x = 2, w = 3, b = 1$$

$$y = 15, \eta = 0.1$$

*Forward Pass* (Proses Prediksi Model) adalah proses mengalirkan *input* dari awal jaringan hingga menghasilkan *output*. Pada tahap ini, belum ada perbaikan bobot, model hanya melakukan prediksi berdasarkan bobot saat ini. Model linear sederhana menghitung *output* dengan mengalikan *input* dan bobot, lalu menambahkan bias.
Rumus

$$\hat{y} = w \cdot x + b \tag{29}$$

Perhitungan

$$\hat{y} = (3 \times 2) + 1 = 7$$

Perhitungan *Loss* (Mengukur Besarnya Kesalahan)
*Loss* digunakan untuk mengukur seberapa jauh prediksi model dari nilai target. Semakin besar loss, semakin besar kesalahan model.
Pada contoh ini digunakan *Mean Squared Error* (MSE) karena sederhana dan mudah dipahami.
Rumus

$$\mathcal{L} = (\hat{y} - y)^2$$

Perhitungan

$$\mathcal{L} = (7 - 15)^2 = (-8)^2 = 64$$

*Backpropagation* (Perhitungan Gradien)
*Backpropagation* adalah proses menghitung seberapa besar kontribusi setiap parameter (bobot dan bias) terhadap nilai *loss*.

a. Gradien terhadap *Output*
Langkah pertama adalah menghitung perubahan *loss* terhadap *output* model. Ini menunjukkan bagaimana perubahan *output* memengaruhi *loss*.
Rumus

$$\frac{\partial \mathcal{L}}{\partial \hat{y}} = 2(\hat{y} - y) \tag{30}$$

Perhitungan

$$\frac{\partial \mathcal{L}}{\partial \hat{y}} = 2(7 - 15) = -16$$

b. *Gradien* terhadap Bobot
Tahap berikutnya adalah menghitung seberapa besar bobot memengaruhi *output*, lalu dikaitkan dengan *loss* $\frac{\partial \mathcal{L}}{\partial \mathrm{w}}$ *gradian loss*
Rumus

$$\frac{\partial \hat{\mathrm{y}}}{\partial \mathrm{w}} = \mathrm{x} \tag{31}$$

$$\frac{\partial \mathcal{L}}{\partial \mathrm{w}} = \frac{\partial \mathcal{L}}{\partial \hat{\mathrm{y}}} \times \frac{\partial \hat{\mathrm{y}}}{\partial \mathrm{w}} \tag{32}$$

Perhitungan

$$\frac{\partial \mathcal{L}}{\partial \mathrm{w}} = -16 \times 2 = -32$$

*Update* Bobot dan Bias (*Gradient Descent*) Setelah *gradien* diketahui, bobot dan bias diperbarui menggunakan *learning rate* agar perubahan tidak terlalu besar.
Bobot Baru
Rumus

$$w_{\text{baru}} = w - \eta \cdot \frac{\partial \mathcal{L}}{\partial w} \tag{33}$$

Perhitungan

$$w_{\text{baru}} = 3 - (0.1 \times -32) = 6.2$$

*Bias* Baru
Rumus

$$b_{\text{baru}} = b - \eta \cdot \frac{\partial \mathcal{L}}{\partial b}$$

Perhitungan

$$b_{\text{baru}} = 1 - (0.1 \times -16) = 2.6$$

*Loss* Baru (Evaluasi Akhir)
Rumus

$$\mathcal{L}_{\text{baru}} = (\hat{y}_{\text{baru}} - y)^2$$

Perhitungan

$$\mathcal{L}_{\text{baru}} = (15 - 15)^2 = 0$$

*Loss* menjadi 0, menandakan model telah belajar dengan baik.

**11. *Update* Bobot (Tahap Akhir *Training*)**

Setelah proses *forward pass*, perhitungan *loss*, dan *backpropagation* selesai dilakukan, model memasuki tahap akhir dalam satu siklus pelatihan, yaitu *update* bobot. Pada tahap ini, informasi gradien yang telah dihitung digunakan untuk menyesuaikan parameter jaringan agar kesalahan prediksi pada iterasi berikutnya menjadi lebih kecil.

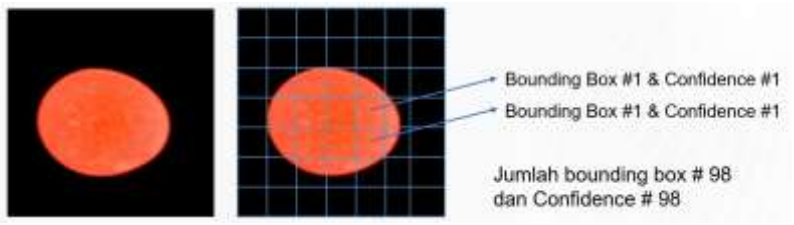
Rumus *Update* Bobot Secara umum, proses *update* bobot dapat dituliskan sebagai:

$$W_{t+1} = W_t - \eta \cdot \nabla \mathcal{L} \quad (34)$$

Keterangan:
$W_t$: Bobot sebelum diperbarui
$W_{t+1}$: Bobot setelah diperbarui
$\eta$: *Learning rate*
$\nabla \mathcal{L}$: Gradien *loss* terhadap bobot
Rumus ini menunjukkan bahwa bobot diperbarui dengan bergerak berlawanan arah gradien, sehingga nilai loss semakin kecil.

### *Head* Mode *Inference* (*Decoding Bounding Box*)

Pada mode *inference*, blok *head* berfungsi untuk mendekode fitur yang dihasilkan *backbone* dan *neck* menjadi prediksi *bounding box* dan *confidence score*. Meskipun pada mode *inference* model tidak menghitung *loss* atau memerlukan *ground truth*, alur proses di *head* secara struktural sama dengan mode *training*. Artinya, *feature maps* dari *backbone* dan *neck* tetap diproses melalui layer layer prediksi yang sama, menghasilkan koordinat *bounding box* serta probabilitas kelas objek. Perbedaannya terletak pada tidak adanya backpropagation; model hanya melakukan *forward pass* untuk menghasilkan *output* akhir. Selain itu, pada tahap post processing, prediksi *bounding box* di mode *inference* biasanya langsung di saring menggunakan teknik *Non Maximum Suppression* (NMS) untuk menghilangkan kotak yang tumpang tindih, sehingga *output* akhir siap digunakan untuk deteksi objek tanpa perlu evaluasi *loss*. Secara matematis, *head* menghasilkan dua *output* utama yaitu Prediksi koordinat *bounding box* dan Prediksi probabilitas kelas berikut ini contoh Gambar 30 Prediksi koordinat *bounding box*:




**Gambar 30. Gambar Prediksi koordinat *bounding box***

**a. Proses deteksi pada Detect Head**

**1. Output Conv2D Detect Head**

pada Detect Head Pada YOLOv8, layer Detect Head menghasilkan output untuk setiap feature map (P3, P4, P5). Output tersebut digunakan untuk memprediksi bounding box dan klasifikasi objek. Regresi Bounding Box Output regresi memiliki bentuk tensor:

$$[B,\ 4 \times reg\_max,\ H,\ W]$$

Keterangan:
B = batch size
reg_max = jumlah distribusi jarak bbox
H, W = ukuran feature map
Contoh pada feature map P3:
reg_max = 4
ukuran P3 = 80 × 80
Sehingga bentuk tensor: [1,16,80,80]
karena

$$4 \times 4 = 16$$

Output ini berisi nilai logit mentah untuk jarak bounding box. Pengambilan Satu Titik Grid Sebagai contoh perhitungan, diambil satu titik pada feature map P3:
Grid: (i = 5, j = 5)
Stride: 8
Output Conv2D regresi pada titik tersebut:

**Tabel 2. Nilai logit**

| Sisi | Logit |
|---|---|
| LEFT | [1.0, 2.0, 4.0, 1.0] |
| TOP | [0.5, 1.5, 3.0, 1.0] |
| RIGHT | [0.2, 2.5, 1.0, 0.3] |
| BOTTOM | [0.1, 1.0, 3.5, 0.4] |

Nilai tersebut merupakan output mentah sebelum diproses menggunakan softmax dan Distribution Focal Loss (DFL).

**2. Perhitungan Softmax dan Distribution Focal Loss (DFL)**

Karena YOLOv8 menggunakan Distribution Focal Loss, setiap sisi bounding box dihitung sebagai distribusi probabilitas menggunakan fungsi softmax. Pada sisi LEFT Logit: [ 1.0,2.0,4.0,1.0]

**Tabel 3. nilai eksponensial**

| **Logit** | $e^{logit}$ |
|---|---|
| 1.0 | 2.71 |
| 2.0 | 7.39 |
| 4.0 | 54.60 |
| 1.0 | 2.71 |

Jumlah:

$$2.71 + 7.39 + 54.60 + 2.71 = 67.41$$

Probabilitas softmax: [0.04, 0.11, 0.81, 0.04]
Perhitungan Ekspektasi Jarak
Jarak dihitung menggunakan nilai ekspektasi:

$$distance = \sum index \times probability$$

Untuk sisi LEFT:

$$0 \times 0.04 + 1 \times 0.11 + 2 \times 0.81 + 3 \times 0.04 = 1.88$$

**Tabel 4. Hasil jarak dari setiap sisi**

| Sisi | Jarak |
|---|---|
| LEFT | 1.88 |
| TOP | 1.83 |
| RIGHT | 1.23 |
| BOTTOM | 1.91 |

Nilai ini masih dalam satuan grid.

**3. Pembentukan Bounding Box pada Gambar Asli**

Menentukan Titik Pusat Grid

$$cx = (j + 0.5) \times stride$$
$$cy = (i + 0.5) \times stride$$

Dengan:
$i = 5$
$j = 5$
$stride = 8$
Maka:

$$cx = 44$$
$$cy = 44$$

Mengubah Jarak ke Piksel Jarak dikalikan dengan stride = 8.

**Tabel 5. Perhitungan jarak pixel**

| Sisi | Perhitungan | Pixel |
|---|---|---|
| LEFT | 1.88 × 8 | 15.04 |
| TOP | 1.83 × 8 | 14.64 |
| RIGHT | 1.23 × 8 | 9.84 |
| BOTTOM | 1.91 × 8 | 15.28 |

Menghitung Koordinat Bounding Box Koordinat kiri atas:

$$x_1 = cx - left$$
$$x_1 = 44 - 15.04 = 28.96$$
$$y_1 = cy - top$$
$$y_1 = 44 - 14.64 = 29.36$$

Koordinat kanan bawah:

$$x_2 = cx + right$$
$$x_2 = 44 + 9.84 = 53.84$$
$$y_2 = cy + bottom$$
$$y_2 = 44 + 15.28 = 59.28$$

Hasil Akhir Bounding Box prediksi:

(x1, x2, y1, y2)

(28.96, 29.36, 53.84, 59.28)

Koordinat ini membentuk bounding box objek pada gambar asli yang diprediksi dari satu titik grid pada feature map.

**b. Proses Klasifikasi pada Detect Head YOLOv8**

**1. Klasifikasi Kelas (Class Prediction)**

Pada arsitektur YOLOv8, Detect Head tidak hanya memprediksi bounding box, tetapi juga melakukan klasifikasi objek untuk menentukan jenis objek yang terdeteksi.

Proses klasifikasi dilakukan menggunakan layer Conv2D klasifikasi yang menghasilkan class logits untuk setiap grid pada feature map.

Bentuk output klasifikasi adalah:

$$cls\_logits = (nc \times H \times W)$$

Keterangan:

nc: jumlah kelas objek

H, W: tinggi dan lebar feature map

cls_logits: nilai prediksi mentah (logit) untuk setiap kelas

Conv2D klasifikasi biasanya memiliki konfigurasi: Conv2D (1×1, out_channels = nc) Layer ini mengubah feature map menjadi skor prediksi untuk setiap kelas pada setiap posisi grid.

**2. Definisi Kelas pada Dataset**

Pada penelitian ini digunakan **dua kelas objek**, yaitu:


**Gambar 31. Kelas objek**

**3. Prediksi Klasifikasi dan Confidence Score**

Sebagai contoh, pada grid (i = 5, j = 5), Detect Head menghasilkan nilai class logits:

$$cls\_logits = [2.5, -0.7]$$

Nilai logit ini kemudian diubah menjadi probabilitas menggunakan fungsi sigmoid.
Fungsi sigmoid:

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

**Tabel 6. Hasil perhitungan**

| Kelas | Logit | Probabilitas |
|---|---|---|
| telur utuh | 2.5 | ≈ 0.92 |
| telur retak | -0.7 | ≈ 0.33 |

**4. Interpretasi Hasil Prediksi**

Nilai probabilitas menunjukkan tingkat keyakinan (confidence) model terhadap masing-masing kelas.
Pada contoh ini:

Telur utuh = 0.92
Telur retak = 0.33

Karena nilai probabilitas tertinggi adalah 0.92, maka objek pada grid tersebut diprediksi sebagai telur utuh.

**a. Feature Map (1 Grid)**

Feature map yang dihasilkan dari proses ekstraksi fitur:

$$f = [0.8, -1.2, 0.5]$$

Layer Conv2D digunakan untuk menghitung class logits pada setiap grid. Perhitungan dilakukan untuk setiap kelas dengan menggunakan bobot hasil training.

**b. Bobot Conv2D (Hasil Training)**

Kelas 0 (Telur Utuh)

$$w_0 = [1.5, -0.8, 0.3]$$
$$b_0 = 0.2$$

Kelas 1 (Telur Retak)

$$w_1 = [-0.6, 1.2, -0.4]$$
$$b_1 = -0.1$$

Perhitungan Logit
Logit Kelas 0

$$(0.8 \times 1.5) + (-1.2 \times -0.8) + (0.5 \times 0.3) + 0.2$$
$$= 1.2 + 0.96 + 0.15 + 0.2$$
$$= 2.51$$

Logit Kelas 1

$$(0.8 \times -0.6) + (-1.2 \times 1.2) + (0.5 \times -0.4) - 0.1$$

$$= -0.48 - 1.44 - 0.2 - 0.1$$
$$\approx -0.7$$

Sigmoid Confidence per Kelas

YOLOv8 tidak menggunakan softmax untuk klasifikasi kelas, tetapi menggunakan fungsi sigmoid untuk menghitung confidence setiap kelas secara independen.

Rumus fungsi sigmoid:

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

Perhitungan Confidence

Kelas 0 (Telur Utuh)

$$\sigma(2.5) \approx 0.92$$

Kelas 1 (Telur Retak)

$$\sigma(-0.7) \approx 0.33$$

Hasil Prediksi

Karena nilai confidence kelas Telur Utuh (0.92) lebih tinggi dibandingkan kelas Telur Retak (0.33), maka model memprediksi objek sebagai Telur Utuh.

**5. Penentuan Kelas dan Hasil Prediksi Objek**

**a. Menentukan Kelas yang Diprediksi**

Setelah nilai probabilitas untuk setiap kelas diperoleh dari proses sigmoid, langkah berikutnya adalah menentukan kelas yang memiliki nilai probabilitas tertinggi.

Probabilitas yang dihasilkan:

$$[0.92,\ 0.33]$$

Penentuan kelas dilakukan menggunakan fungsi argmax, yaitu memilih indeks dengan nilai terbesar.

$$kelas\_pred = argmax([0.92,\ 0.33])$$

Hasilnya:

kelas = 0

Berdasarkan definisi kelas pada dataset Label 0 untuk kelas telur utuh dan 1 untuk kelas telur retak Maka objek tersebut diprediksi sebagai telur utuh.

**b. Menentukan Confidence Score**

Confidence menunjukkan tingkat keyakinan model terhadap prediksi kelas. Nilai confidence diambil dari probabilitas terbesar.

$$confidence = max([0.92,\ 0.33])$$

Sehingga diperoleh:

$$confidence = 0.92$$

Artinya model memiliki tingkat keyakinan sebesar 92% bahwa objek tersebut adalah telur utuh.

**c. Menggabungkan Prediksi Kelas dengan Bounding Box**

Bounding box yang telah dihitung sebelumnya memiliki format:

$$bbox = (x_1,\ y_1,\ x_2,\ y_2)$$

Hasil bounding box:

$$(28.96,\ 29.36,\ 53.84,\ 59.28)$$

Koordinat tersebut menunjukkan:

(x1, y1): titik kiri atas bounding box

(x2, y2): titik kanan bawah bounding box

**d. Hasil Akhir Prediksi dari Satu Grid**

Berdasarkan hasil perhitungan bounding box dan klasifikasi, maka prediksi objek dari satu grid adalah:

Bounding Box (28.96, 29.36, 53.84, 59.28)

Kelas telur utuh

Confidence 0.92

**Gambar 32. Hasil prediksi**

### 5. Validasi Model

Validasi model adalah proses evaluasi kinerja model pada *dataset* yang tidak digunakan saat pelatihan (*validation* set). Tujuannya untuk menilai kemampuan generalisasi model, mendeteksi *overfitting*, dan memilih model terbaik berdasarkan metrik evaluasi seperti akurasi, IoU, atau *loss*. Proses ini penting agar model yang dihasilkan tidak hanya cocok dengan data latih, tetapi juga mampu bekerja baik pada data baru atau citra nyata.

### *6.* Penyetelan Hyperparameter

Penyetelan *hyperparameter* (*hyperparameter tuning*) adalah langkah untuk menemukan kombinasi parameter optimal yang mengontrol perilaku model, seperti *learning rate*, jumlah layer, ukuran *batch* dan threshold deteksi. Penyetelan dilakukan melalui eksperimen, *grid search*, *random search*, atau metode otomatis, untuk meningkatkan kinerja model sekaligus mencegah *overfitting* atau *underfitting*.

**Tabel 7. *Hyperparameter***

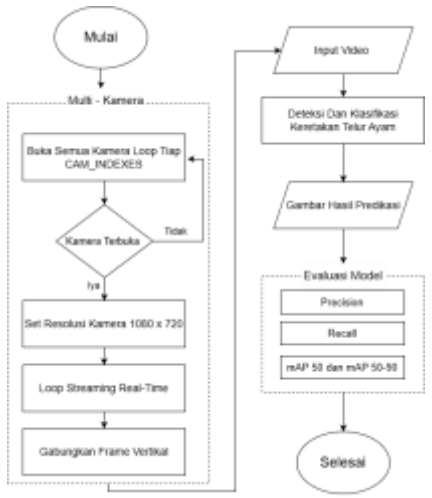
| *Hyperparameter* | Deskripsi | Nilai *Default* / Umum |
|---|---|---|
| *Learning Rate (lr)* | Kecepatan pembaruan bobot jaringan selama *training*. | 0.001 |
| *Batch Size* | Jumlah citra yang diproses sebelum bobot diperbarui. | 16 |
| *Epochs* | Jumlah siklus pelatihan penuh pada seluruh *dataset*. | 100 |
| *Image Size* | Resolusi *input* citra yang dipakai oleh model. | 640×640 |
| *IoU Threshold* | Ambang batas IoU untuk menentukan *true positive* saat NMS (*No Max Suppression*). | 0.7 |
| *Confidence Threshold* | Ambang probabilitas minimum untuk memutuskan apakah prediksi dianggap valid. | 0.5 |

### *7.* Model deteksi dan klasifikasi

Model adalah struktur jaringan yang digunakan untuk deteksi dan klasifikasi objek, termasuk arsitektur *backbone*, *neck*, dan *head*. Dalam konteks klasifikasi keretakan telur ayam, model menerima citra sebagai *input*, mengekstraksi fitur, memprediksi kelas (Telur Utuh atau Telur Retak), dan memprediksi posisi *bounding box*. Model dapat dilatih menggunakan *dataset* berlabel, dan hasilnya di evaluasi melalui *loss*, akurasi, dan IoU untuk memastikan prediksi yang akurat.

### 2.4.3 Testing

Gambar alur testing ini menggambarkan tahapan pengujian sistem secara menyeluruh, mulai dari proses inisialisasi, pengambilan data dari kamera, hingga penampilan hasil. Alur ini menunjukkan bagaimana sistem membaca *input*, memproses data, dan menampilkan keluaran tanpa melibatkan proses pelatihan. Dengan adanya alur testing ini, diharapkan pembaca dapat memahami proses kerja sistem saat diuji dalam kondisi operasional sebenarnya.




**Gambar 33. Tahapan Testing**

#### 1. Data Multi-Kamera

Merupakan tahap awal dalam proses testing sistem, di mana beberapa kamera diaktifkan secara bersamaan untuk memperoleh data citra atau video sebagai masukan sistem. Berdasarkan diagram alur, tahap ini ditunjukkan setelah proses Mulai dan berfungsi sebagai sumber utama data pengujian.

Pada tahap ini, setiap kamera di inisialisasi dan di konfigurasi sesuai parameter yang telah ditentukan, seperti indeks perangkat dan resolusi citra. Sistem memastikan bahwa seluruh kamera dapat terbuka dan beroperasi dengan baik sebelum proses selanjutnya dijalankan. Data yang diambil dari masing-masing kamera bersifat *real time* dan merepresentasikan kondisi lingkungan sebenarnya saat proses testing berlangsung.

Penggunaan *input* data multi-kamera bertujuan untuk memperoleh sudut pandang yang lebih luas dan komprehensif terhadap objek yang diamati. Dengan demikian, sistem dapat melakukan pengujian kinerja secara lebih optimal, baik dari sisi kestabilan pengambilan data maupun kemampuan model dalam memproses dan menghasilkan prediksi dari berbagai sumber citra secara simultan.

#### 2. Masukkan *Library* (cv2, NumPy, CAM INDEXES, Resolusi W × H)

Tahap ini merupakan proses inisialisasi awal sistem dengan memanggil pustaka (*library*) yang dibutuhkan. *Library* OpenCV (cv2) digunakan untuk mengakses dan mengelola perangkat kamera, NumPy dimanfaatkan untuk pengolahan data citra, khususnya dalam penggabungan *frame*, sedangkan CAM_INDEXES digunakan untuk mendefinisikan indeks kamera yang akan diaktifkan. Selain itu, parameter

resolusi lebar (W) dan tinggi (H) ditentukan untuk mengatur kualitas citra yang dihasilkan oleh masing-masing kamera.

**3. Buka Semua Kamera (*Loop* Tiap CAM INDEXES)**

Pada tahap ini, sistem membuka seluruh kamera yang telah di definisikan pada variabel CAM_INDEXES. Proses dilakukan menggunakan perulangan (*loop*) sehingga setiap kamera di akses satu per satu. Setiap kamera yang berhasil dibuka akan disimpan ke dalam sebuah daftar (*list*) untuk digunakan pada proses selanjutnya. Tahapan ini memastikan bahwa sistem mampu menangani lebih dari satu sumber *input* secara simultan.

**4. Kamera Terbuka**

Setelah proses pembukaan kamera dilakukan, sistem melakukan pengecekan untuk memastikan bahwa setiap kamera berhasil terbuka dan dapat di akses. Apabila salah satu kamera gagal dibuka, proses testing akan dihentikan untuk mencegah terjadinya kesalahan pada tahap berikutnya. Tahap ini berfungsi sebagai validasi awal sebelum sistem memasuki proses pengambilan data.

**5. Set Resolusi Kamera (1080 × 720)**

Setelah kamera dipastikan terbuka, sistem mengatur resolusi masing-masing kamera menjadi 1080 × 720 piksel. Pengaturan resolusi ini bertujuan untuk menyeragamkan ukuran *frame* yang dihasilkan, sehingga memudahkan proses pengolahan dan penggabungan citra. Resolusi yang konsisten juga membantu menjaga kestabilan tampilan selama proses berlangsung.

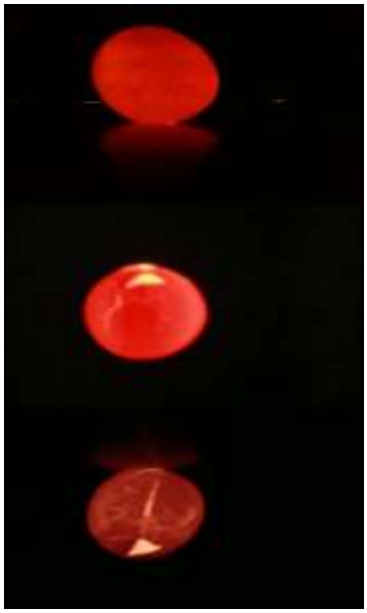
**Gambar 34. Resolusi Kamera 1080 x 720**

***6. Loop While True***

Tahap ini merupakan proses utama dalam testing sistem, di mana sistem berjalan secara kontinu menggunakan perulangan *while true*. Pada setiap iterasi, sistem membaca *frame* dari seluruh kamera secara *real time*. Proses ini memungkinkan tampilan video ditampilkan secara langsung dan berkelanjutan selama sistem aktif, hingga pengguna menghentikan proses secara manual atau terjadi kegagalan pembacaan kamera. Sehingga dilakukan perekaman secara langsung.

**7. Menggabungkan *Frame* Vertikal**

*Frame* yang berhasil dibaca dari masing-masing kamera kemudian digabungkan secara vertikal menjadi satu tampilan satu. Proses penggabungan ini dilakukan menggunakan fungsi pengolahan *array* dari *NumPy*. Hasil penggabungan tersebut menampilkan seluruh citra dari kamera atas, tengah, dan bawah dalam satu *frame*, sehingga memudahkan proses pemantauan dan evaluasi sistem selama tahap testing.



**Gambar 35. Penggabungan *Frame***

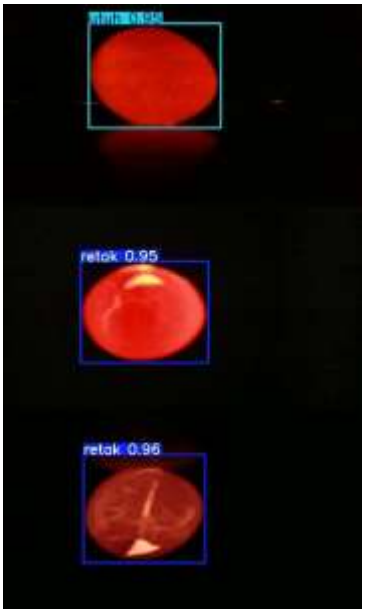
a. Model Deteksi dan Klasifikasi Keretakan telur ayam

Model deteksi dan klasifikasi keretakan telur ayam berfungsi untuk menganalisis citra dari sistem multi-kamera dengan mendeteksi keberadaan telur serta mengklasifikasikan kondisinya berdasarkan ada atau tidaknya keretakan pada cangkang. Pada tahap testing, model dijalankan dalam mode inferensi menggunakan bobot hasil pelatihan tanpa pembaruan parameter. Citra hasil tangkapan kamera terlebih dahulu di praproses agar sesuai dengan format *input* model. Selanjutnya, model mengekstraksi fitur visual pada permukaan telur dan menghasilkan keluaran berupa lokasi telur (*bounding box*) serta label klasifikasi, seperti telur normal atau telur retak. Hasil prediksi ditampilkan pada citra untuk memudahkan observasi dan menjadi dasar evaluasi kinerja sistem. *Input* gambar testing pada YOLOv8 merupakan citra hasil tangkapan sistem multi-kamera yang digunakan sebagai masukan model pada tahap pengujian (*inference*). Citra ini terlebih dahulu dipraproses melalui penyesuaian ukuran, normalisasi nilai piksel, dan pengaturan format *input* agar sesuai dengan kebutuhan model. Selanjutnya, citra diproses oleh *backbone* YOLOv8 untuk mengekstraksi fitur visual penting pada permukaan telur, yang kemudian di fusikan pada bagian *neck* guna menangkap informasi objek dari berbagai skala. Pada tahap *detection head*, YOLOv8 secara langsung memprediksi koordinat *bounding box*, skor kepercayaan, dan label kelas secara *anchor free* tanpa menggunakan *anchor box*. Hasil prediksi kemudian di filter menggunakan *confidence threshold* dan *Non Maximum Suppression* (NMS) untuk menghasilkan deteksi akhir yang akurat, yang selanjutnya di visualisasikan sebagai dasar evaluasi kinerja model dalam mendeteksi dan mengklasifikasikan keretakan telur ayam.

b. *Output* Deteksi dan Klasifikasi Keretakan telur ayam

*Output* deteksi dan klasifikasi merupakan hasil akhir dari proses inferensi model YOLOv8 yang ditampilkan dalam bentuk visual pada citra *input*. *Output* ini meliputi *bounding box* yang menunjukkan lokasi telur, label kelas yang mengindikasikan kondisi telur (normal atau retak), serta nilai *confidence* yang merepresentasikan tingkat keyakinan model terhadap hasil prediksi. Seluruh hasil deteksi yang

ditampilkan telah melalui proses penyaringan menggunakan *confidence threshold* dan *Non Maximum Suppression* (NMS) untuk menghilangkan prediksi yang tidak relevan dan tumpang tindih. Visualisasi hasil ini memudahkan pengguna dalam melakukan pemantauan secara *real time* serta menjadi dasar dalam mengevaluasi kinerja sistem deteksi dan klasifikasi keretakan telur ayam.




**Gambar 36. *Output* Deteksi dan Klasifikasi Keretakan Telur Ayam**

c. Evaluasi Model

Evaluasi kinerja model dilakukan untuk menilai kemampuan sistem dalam mendeteksi dan mengklasifikasikan keretakan telur ayam berdasarkan hasil *output* yang dihasilkan pada tahap testing. Evaluasi ini dilakukan dengan menganalisis kesesuaian antara hasil prediksi model dan kondisi objek sebenarnya, baik dari segi ketepatan lokasi deteksi (*bounding box*) maupun ketepatan klasifikasi kelas (telur normal atau telur retak). Parameter evaluasi yang digunakan meliputi tingkat akurasi deteksi, nilai *confidence*, serta kestabilan model dalam menghasilkan prediksi secara konsisten pada berbagai kondisi pengujian. Hasil evaluasi ini digunakan sebagai dasar untuk menilai efektivitas model dan menentukan kebutuhan penyempurnaan sistem pada tahap pengembangan selanjutnya.

***Precision*** menunjukkan seberapa banyak prediksi positif model yang benar. Metrik ini mengukur tingkat kesalahan *false positive*, yaitu saat model mendeteksi telur retak padahal sebenarnya tidak retak.

Rumus:

$$\text{Precision} = \frac{TP}{TP + FP}$$

Keterangan:

TP (*True Positive*): Telur retak terdeteksi sebagai retak

FP (*False Positive*): Telur normal terdeteksi sebagai retak

Contoh Perhitungan:

TP = 45

FP = 5

$$\text{Precision} = \frac{45}{45+5} = \frac{45}{50} = 0{,}90$$

Interpretasi:
*Precision* sebesar 0,90 menunjukkan bahwa 90% prediksi telur retak oleh model adalah benar.

***Recall*** menunjukkan kemampuan model dalam menemukan seluruh objek yang seharusnya terdeteksi. Metrik ini berfokus pada kesalahan *false negative*, yaitu saat telur retak tidak terdeteksi.
Rumus:

$$\text{Recall} = \frac{TP}{TP+FN}$$

Keterangan:
FN (*False Negative*): Telur retak tetapi tidak terdeteksi
TP (*True Positive*): Telur retak terdeteksi sebagai retak
Contoh Perhitungan:
TP = 45
FN = 10

$$\text{Recall} = \frac{45}{45+10} = \frac{45}{55} = 0{,}82$$

Interpretasi:
*Recall* sebesar 0,82 menunjukkan bahwa model berhasil mendeteksi 82% dari seluruh telur yang benar-benar retak.

***Mean Average Precision* (mAP)** merupakan metrik utama pada YOLOv8 untuk menilai performa keseluruhan model deteksi, yang dihitung dari rata-rata *Average Precision (AP)* pada seluruh kelas dan berbagai *threshold* IoU.
Rumus umum:

$$mAP = \frac{1}{N}\sum_{i=1}^{N} A\,P_i$$

Contoh Perhitungan:
AP kelas telur utuh = 0,88
AP kelas telur retak = 0,84

$$mAP = \frac{0{,}88 + 0{,}84}{2} = 0{,}86$$

Interpretasi:
Nilai mAP sebesar 0,86 menunjukkan bahwa model memiliki performa deteksi dan klasifikasi yang baik secara keseluruhan.

## 2.6 Waktu dan Lokasi Penelitian

Penelitian ini dimulai pada bulan Februari 2025 – Januari 2026. Penelitian dilaksanakan diLaboratorium Kecerdasan Buatan, Departemen Teknik Informatika Universitas Hasanuddin, Jalan Poros Malino, Kelurahan Borongloe, Kecamatan Bontomarannu, Kabupaten Gowa, Provinsi Sulawesi Selatan.