BAB II
TINJAUAN PUSTAKA

2.1 Penelitan-Penelitian Sebelumnya

Salah satu inspirasi yang mendasari penelitian ini adalah Edblocks, sebuah platform pemrograman blok yang dirancang untuk mendidik anak-anak dalam belajar pemrograman dasar. Edblocks menggunakan pendekatan visual dan interaktif, memungkinkan pengguna untuk menyusun blok-blok kode secara digital untuk menggerakkan robot[3]. Konsep ini telah membuka jalan bagi penelitian ini, di mana ide utamanya adalah mengembangkan sistem yang memungkinkan pengguna menyusun blok program dalam dunia nyata, seperti menyusun *puzzle*, dan kemudian mengenali susunan blok tersebut melalui teknologi deteksi objek. Dengan demikian, penelitian ini berusaha mengambil langkah lebih jauh dari pendekatan Edblocks, dengan mengintegrasikan interaksi fisik dan digital dalam proses belajar pemrograman.

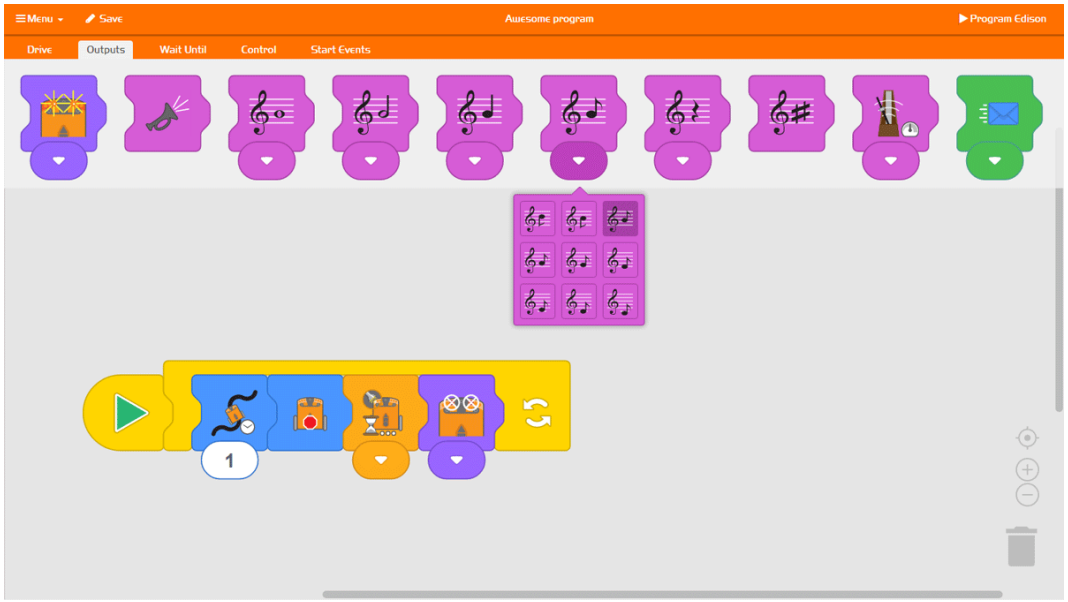
Terdapat penelitian lainnya telah yang menggunakan teknologi pemrosesan citra dalam robot industri untuk mendeteksi dan mengurutkan objek dapat meningkatkan efisiensi dan akurasi dalam proses produksi[4]. Dalam penelitian tersebut, lengan robot cerdas dilengkapi dengan kamera Pi untuk akuisisi gambar dan deteksi objek yang kemudian dikirim ke raspberry Pi untuk melakukan berbagai fungsi pemrosesan citra seperti pra-pemrosesan (rotasi gambar, pengubahan ukuran, konversi skala abu-abu) untuk memanipulasi gambar dan ekstraksi fitur untuk mengekstrak informasi yang diperlukan dari gambar.

Penelitian lainnya yaitu adalah tentang You Only Look Once (YOLO) yang diperkenalkan oleh Joseph Redmon. Penelitian ini mengusulkan sebuah pendekatan baru untuk deteksi objek yang dapat memprediksi kotak pembatas dan probabilitas kelas objek secara langsung dari gambar penuh dalam satu evaluasi. Penelitian ini menggunakan jaringan saraf tiruan konvolusional untuk melakukan deteksi objek

sebagai masalah regresi. Penelitian ini juga menunjukkan bahwa YOLO memiliki kecepatan dan akurasi yang tinggi dalam mendeteksi objek dalam waktu singkat[5].

2.2 Pemrograman Blok

Pemrograman blok, juga dikenal sebagai pemrograman berbasis blok merupakan salah satu jenis bahasa pemrograman visual yang memanfaatkan elemen visual untuk membangun program. Dalam pendekatan ini, pengguna dapat "membangun" program dengan menyusun blok kode yang mewakili berbagai fungsi, struktur kontrol, dan konsep-konsep pemrograman[6], seperti yang diilustrasikan pada Gambar 2.1.



Gambar 2. 1 Ilustrasi pemrograman blok

Gambar 2.1 menggambarkan bahwa, dengan menggunakan blok-blok yang dapat disusun seperti *puzzle*, pengguna dapat membuat program tanpa harus mengetik kode. Pendekatan ini memfasilitasi pemahaman tentang bagaimana program bekerja dan membantu pengguna, terutama mereka yang baru memulai atau tidak memiliki latar belakang teknis, untuk belajar pemrograman dengan cara yang lebih mudah, intuitif, dan menyenangkan[7].

Pemrograman blok telah terbukti efektif dalam mengajarkan pemrograman dan pemikiran komputasional kepada siswa. Dengan memberikan pengalaman belajar yang interaktif dan menarik. Selain itu, pemrograman blok juga dapat membantu siswa dalam mengembangkan keterampilan pemecahan masalah dan pemikiran algoritma[8]. Keuntungan utama dari pemrograman blok adalah kemampuannya untuk mengurangi kesalahan sintaks dan mempercepat proses pembuatan atau penulisan kode program. Yang mana pada penelitian ini adalah membantu pengguna untuk mengenalkan pemrograman untuk menggerakkan robot dengan mudah.

Meski banyak manfaatnya pendekatan ini juga memiliki keterbatasan yang mungkin dihadapi, antara lain:

1. Keterbatasan Kompleksitas

   Pemrograman blok biasanya dirancang untuk pemula dan proyek-proyek skala kecil hingga menengah. Untuk proyek yang lebih kompleks dan skala besar, pemrograman blok mungkin tidak cukup kuat atau fleksibel. Misalnya, pendekatan ini tidak akan mencangkup fitur-fitur lanjutan seperti multithreading atau pemrograman berorientasi objek dengan cara yang sama seperti bahasa pemrograman teks.
2. Keterbatasan dalam Belajar Sintaksis

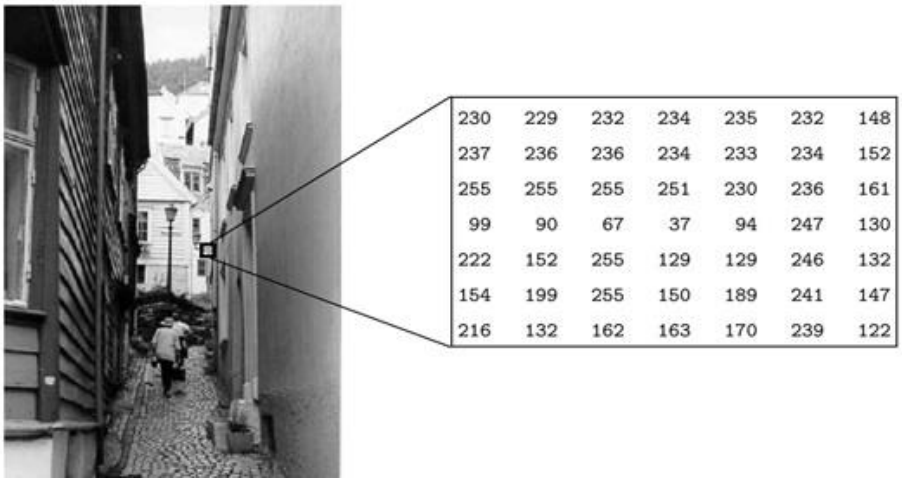
   Meskipun pemrograman blok dapat membantu pemula memahami konsep dasar pemrograman, mereka tidak mengajarkan sintaksis bahasa pemrograman tertentu. Ini bisa menjadi masalah ketika pemrogram ingin beralih ke bahasa pemrograman teks, karena mereka harus belajar sintaksis dari awal.
3. Keterbatasan dalam *Debugging*

   Pemrograman blok dapat membuat proses *debugging* lebih sulit. Ini karena kebanyakkan pemrograman blok sudah dirancang agar tidak terjadi eror pada kode. Sedangkan dalam pemrograman teks, programmer dapat melihat dan memodifikasi setiap bagian kode.

2.3 Citra Digital

Citra digital adalah representasi visual dari suatu objek yang dibentuk oleh susunan matematis piksel. Piksel, atau elemen gambar, adalah unit dasar dari sebuah citra digital, yang masing-masing memegang nilai tertentu yang mencerminkan informasi seperti intensitas cahaya atau warna. Citra digital dapat dihasilkan melalui berbagai cara, seperti dengan kamera digital, *scanner*, atau melalui proses komputasi.

Piksel ini disusun dalam apa yang disebut *array* dua dimensi, sebuah struktur data yang menyimpan nilai dari setiap piksel dan mewakili posisi mereka dalam citra. Nilai yang disimpan ini mencerminkan informasi tentang piksel tersebut, seperti yang digambarkan pada Gambar 2.2.



Gambar 2. 2 Ilustrasi citra digital terbentuk

Seperti Gambar 2.2, dalam citra digital gambar tersebut bisa diibaratkan sebagai kumpulan titik-titik atau piksel. Setiap titik atau piksel ini mewakili bagian kecil dari gambar dan memiliki nilai tertentu yang mencerminkan warna dan intensitas cahaya dari bagian tersebut. Jadi, ketika kita melihat gambar tersebut dalam bentuk digital, kita sebenarnya sedang melihat kumpulan piksel ini yang telah disusun sedemikian rupa sehingga membentuk gambaran keseluruhan.

Dalam penelitian ini, citra digital sangat penting karena berfungsi sebagai jembatan antara blok fisik yang disusun oleh pengguna dan sistem pemrograman blok yang akan dibuat. Setelah pengguna menyusun blok-blok fisik, kamera akan digunakan untuk mengambil citra dari susunan blok tersebut. Citra tersebut kemudian akan diubah menjadi format digital, yang kemudian dapat diproses dan dianalisis oleh program komputer.

2.4 Pengolahan Citra dan Computer Vision

Pengolahan Citra dan *Computer Vision* merupakan dua konsep yang saling terkait dalam bidang analisis dan interpretasi pada sebuah citra digital. Pengolahan Citra berfokus pada manipulasi dan transformasi citra, sedangkan *Computer Vision* bertujuan untuk memahami dan menginterpretasi konten citra. Bagian ini akan menjelaskan kedua konsep ini lebih detail.

2.4.1 Pengolahan Citra

Pengolahan citra, yang biasanya dikenal sebagai *'image processing*' dalam bahasa Inggris, adalah cabang ilmu komputer yang melibatkan serangkaian teknik untuk memanipulasi gambar. Tujuannya adalah untuk meningkatkan kualitas visual, seperti kontras atau kejernihan, dan memfasilitasi ekstraksi informasi yang lebih baik, seperti deteksi tepi atau pengenalan pola.

Proses pengolahan citra umumnya melibatkan tiga tahap utama: pra-pemrosesan, peningkatan, dan ekstraksi fitur. Pra-pemrosesan melibatkan persiapan gambar untuk analisis lebih lanjut, seperti pengurangan *noise*, peningkatan kontras, dan normalisasi. Peningkatan melibatkan manipulasi gambar untuk meningkatkan kualitas visualnya, seperti penajaman, peningkatan kontras, dan penyesuaian warna. Ekstraksi fitur melibatkan identifikasi dan ekstraksi informasi penting dari gambar, seperti segmentasi, deteksi tepi, dan pengenalan pola.

Namun, pengolahan citra terutama teknik dasarnya memiliki beberapa keterbatasan. Seperti pada teknik peningkatan gambar meski dapat meningkatkan

kualitas visual, sering kali tidak mampu sepenuhnya menghilangkan *noise* atau artifak dan dapat mengubah informasi asli dalam gambar, berpotensi mengarah pada kesalahan interpretasi. Teknik ekstraksi fitur, seperti segmentasi, bergantung pada kualitas gambar asli dan mungkin tidak mampu mengidentifikasi fitur dengan akurat jika gambar asli memiliki kualitas rendah atau banyak *noise*.

2.4.2 *Computer Vision*

Computer vision adalah cabang dari ilmu komputer yang berfokus pada pengembangan teknik yang memungkinkan komputer untuk mendapatkan pemahaman tingkat tinggi dari informasi digital[9]. Tujuan dari *computer vision* yaitu untuk menafsirkan, mengubah, dan mendapatkan informasi dari citra digital. Proses ini melibatkan berbagai langkah, termasuk pengenalan pola, segmentasi gambar, dan pengolahan citra, yang bekerja bersama untuk mengidentifikasi dan mengkategorisasi objek dalam citra.

Pengolahan citra berfokus pada manipulasi dan peningkatan kualitas gambar, sedangkan *computer vision* berusaha untuk memahami konten dan konteks dari gambar tersebut. Kedua bidang ini sering bekerja bersama dalam aplikasi seperti pengenalan wajah, navigasi robot, dan analisis medis.

Metode dan teknik dalam *computer vision* melibatkan penerapan algoritma yang canggih seperti model *machine learning*, termasuk *artificial neural network*, untuk menginterpretasi dan menganalisis citra digital. Proses ini mencakup berbagai tugas yang kompleks, seperti segmentasi semantik, di mana setiap piksel dalam citra diklasifikasikan ke dalam kategori yang telah ditentukan, atau deteksi objek, di mana objek yang ada dalam citra diidentifikasi, ditemukan, dan diklasifikasikan[10].

Meskipun *computer vision* menawarkan potensi yang besar dalam berbagai aplikasi, ini bukan berarti tidak lepas dari keterbatasan dan tantangan. Sama seperti dalam pengolahan citra, *computer vision* sangat sensitif terhadap kualitas citra yang dianalisis. Citra dengan resolusi rendah atau yang terdistorsi dapat mengakibatkan

interpretasi yang tidak akurat. Selain itu, menginterpretasi citra yang kompleks atau yang memiliki banyak elemen dapat menjadi tantangan, terutama ketika detail-detail kecil perlu dikenali. Tantangan lainnya melibatkan pelatihan model *machine learning* yang akurat, di mana model itu membutuhkan data pelatihan yang cukup dan variatif, serta sumber daya komputasi yang besar.

2.5 *Artificial Intelligence*

Artificial intelligence atau kecerdasan buatan dapat didefinisikan sebagai cabang ilmu dan teknik yang berfokus pada pembuatan mesin dan program komputer yang cerdas. AI berhubungan erat dengan tugas menggunakan komputer untuk memahami kecerdasan manusia, namun tidak harus membatasi diri pada metode yang dapat diamati secara biologis. Yang dimaksud kecerdasan pada AI ini adalah bagian komputasi dari kemampuan untuk mencapai suatu tujuan di dunia[11]

Seiring waktu, AI telah mengalami perkembangan yang signifikan. Dari awalnya yang berfokus pada tugas-tugas sederhana, AI kini telah berkembang menjadi teknologi yang mampu menangani tugas-tugas yang kompleks dan beragam. Kemajuan ini sebagian besar didorong oleh peningkatan kapasitas komputasi dan ketersediaan data besar, yang memungkinkan AI untuk belajar dan beradaptasi dengan lebih efisien.

Berbagai cabang dari AI telah diidentifikasi dan terus berkembang seiring dengan perkembangan teknologi. Salah satu cabangnya adalah *logical* AI, di mana program ini menggunakan logika matematika untuk membuat keputusan berdasarkan tujuan yang telah ditentukan. Selanjutnya, ada *search* AI yang memeriksa berbagai kemungkinan untuk mencapai solusi, seperti gerakan dalam permainan catur atau inferensi oleh program pembuktian teorema. *Patern recognition* juga menjadi bagian dari AI, di mana program dibuat untuk membandingkan apa yang dilihatnya dengan pola tertentu, seperti mencari wajah dalam suatu adegan[11]. *Computer vision* juga menjadi bagian penting dari AI

terutama pada penelitian ini, di mana program dibuat untuk memahami objek dalam dunia tiga dimensi.

2.6 *Machine Learning*

Machine Learning (ML) adalah cabang dari AI yang berfokus pada pembangunan sistem yang membuat komputer untuk belajar dari data dan membuat keputusan atau prediksi. Dengan menggunakan algoritma dan model yang beragam, ML memungkinkan komputer untuk belajar dari data yang diberikan. Selain itu, metode ini dapat menggunakan pengetahuan yang diperoleh dalam proses pelatihan untuk membuat keputusan atau prediksi yang akurat[12].

Maksud kata belajar pada ML adalah kemampuan sistem untuk secara otomatis memperbaiki kinerja atau prediksi dalam menyelesaikan tugas tertentu berdasarkan pengalaman sebelumnya atau data yang telah diproses. Ini biasanya melibatkan penyesuaian parameter internal sistem berdasarkan data yang diberikan, yang pada akhirnya mempengaruhi bagaimana sistem tersebut merespons data baru di masa mendatang[13].

Perlu diketahui bahwa makna belajar dalam ML berbeda dengan belajarnya manusia. Manusia belajar dengan proses yang lebih kompleks dan melibatkan peningkatan pemahaman, pengetahuan, keterampilan, perilaku, dan nilai-nilai melalui pengamatan, interaksi sosial, dan refleksi pribadi[14]. Proses belajar manusia juga sangat dipengaruhi oleh faktor-faktor seperti motivasi, emosi, dan lingkungan sosial dan fisik. Sebaliknya, ML belajar lebih fokus pada peningkatan kinerja dalam tugas tertentu berdasarkan data. Meskipun istilah belajar digunakan dalam kedua hal tersebut, proses dan hasilnya sangat berbeda antara belajar manusia dengan belajarnya algoritma ML.

ML dapat dikategorikan menjadi tiga jenis utama berdasarkan cara mereka belajar atau dilatih. Berikut merupakan beberapa jenis ML beserta penjelasannya:

1. ***Supervised Learning***: Jenis pelatihan ML di mana algoritma belajar dari data latih yang sudah diberi label. Di sini, istilah label berkaitan dengan hasil

yang diinginkan untuk setiap contoh dalam data latih. Algoritma *supervised learning* menggunakan data latih ini untuk belajar suatu fungsi yang dapat memetakan *input* baru ke output yang sesuai. Tujuannya adalah untuk memperkirakan fungsi pemetaan tersebut dengan seakurat mungkin sehingga dapat membuat prediksi yang tepat untuk data yang belum pernah dilihat sebelumnya[15]. Dalam penelitian ini pendekatan ini digunakan karena data latih (gambar blok perintah) nantinya akan diberi label yang sesuai dengan perintahnya.

2. ***Unsupervised Learning***: Jenis pelatihan di mana algoritma belajar dari data latih yang tidak diberi label. Algoritma *unsupervised learning* mencoba menemukan struktur tersembunyi atau pola dalam data. Teknik ini sering digunakan dalam analisis klaster, di mana tujuannya adalah untuk mengelompokkan *input* yang serupa ke dalam klaster yang sama[16].
3. ***Reinforcement Learning***: Jenis pelatihan di mana algoritma belajar untuk melakukan suatu tugas tertentu dengan cara mencoba berbagai tindakan dan mendapatkan umpan balik dalam bentuk *reward* atau *penalty*[17]. Algoritma *reinforcement learning* belajar dari pengalaman dan mencoba untuk memaksimalkan total *reward* yang diterima melalui serangkaian tindakan.

Untuk mendapatkan model ML yang diharapkan, proses pembuatan model ML melibatkan beberapa langkah penting yang harus diikuti untuk memastikan hasil yang optimal. Langkah-langkah ini dirancang untuk memandu sistem melalui proses pelatihan yang efisien dan efektif, memungkinkan sistem untuk memahami dan memanfaatkan pola dalam data dengan cara yang paling optimal. Berikut merupakan beberapa proses dari ML beserta penjelasannya:

1. **Pengumpulan dan Pembersihan Data:** Langkah pertama dalam proses ML adalah pengumpulan data. Data ini bisa berupa gambar, teks, suara, atau data numerik, tergantung pada masalah yang ingin diselesaikan. Setelah data dikumpulkan, langkah selanjutnya adalah pembersihan data. Pembersihan data melibatkan penghapusan atau koreksi data yang hilang

atau salah, penghapusan duplikat, dan transformasi data ke format yang dapat digunakan oleh model ML.

2. **Pemilihan Model:** Ada banyak model ML yang berbeda, dan pemilihan model yang tepat tergantung pada jenis masalah yang ingin diselesaikan. Beberapa model mungkin bekerja lebih baik untuk klasifikasi, sementara yang lain mungkin lebih cocok untuk regresi atau *clustering*.
3. **Pelatihan Model:** Setelah model dipilih, langkah selanjutnya adalah pelatihan model. Pelatihan model melibatkan penggunaan data latih untuk menyesuaikan parameter model. Tujuannya adalah untuk meminimalkan perbedaan antara prediksi model dan nilai sebenarnya dari data latih.
4. **Evaluasi Model:** Setelah model dilatih, langkah selanjutnya adalah evaluasi model. Evaluasi model melibatkan penggunaan data tes yang tidak digunakan selama pelatihan untuk menguji kinerja model. Metrik evaluasi yang digunakan tergantung pada jenis masalah yang diselesaikan.
5. **Peningkatan Model:** Jika hasil evaluasi model tidak memuaskan, langkah selanjutnya adalah peningkatan model. Peningkatan model dapat melibatkan penyesuaian parameter model, pemilihan fitur ulang, atau bahkan pemilihan model yang berbeda.

ML telah menjadi komponen kunci dalam perkembangan dan kemajuan AI. Dalam beberapa dekade terakhir, ML telah memungkinkan komputer untuk menyelesaikan tugas yang sebelumnya dianggap terlalu kompleks atau tidak mungkin untuk diotomatisasi[18]. Dengan ML, sistem AI dapat belajar dari pengalaman, mengadaptasi respons mereka berdasarkan data baru, dan secara progresif meningkatkan kinerja mereka seiring waktu.

2.7 *Deep Learning*

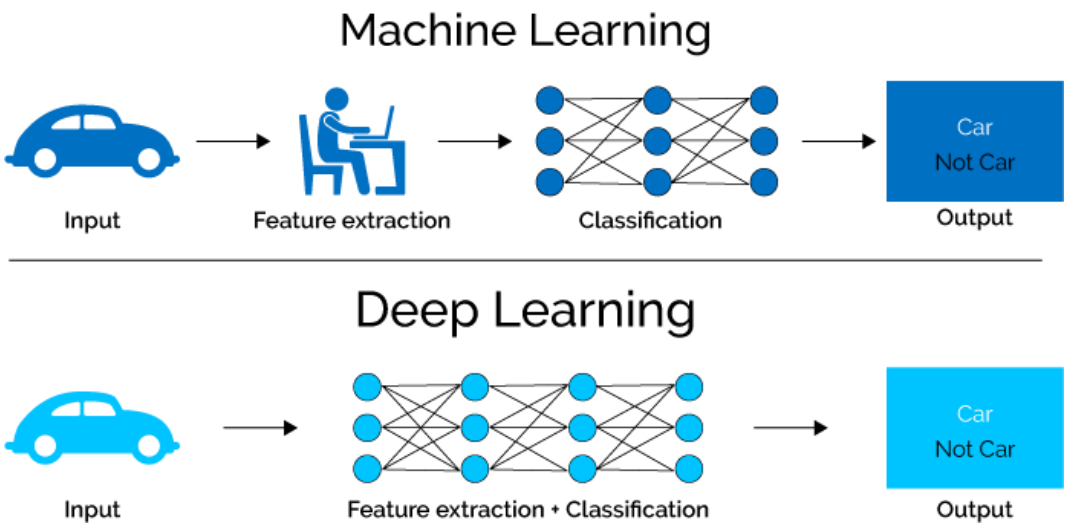
Deep learning (DL) merupakan subkategori dari *Machine learning* (ML), yang secara spesifik berfokus pada implementasi algoritma yang meniru struktur dan fungsi otak manusia, dikenal sebagai *Artificial Neural Networks* (ANN). Konsep ini berakar pada ide bahwa model komputasi dapat dikembangkan

berdasarkan observasi tentang bagaimana otak manusia memproses data, mengidentifikasi pola, dan membuat keputusan[19]. Dengan demikian, DL memungkinkan dilatih dari data yang sangat besar dan kompleks, dan mampu menghasilkan wawasan dan prediksi yang lebih akurat dibandingkan metode tradisional atau teknik ML.

Deep Learning dan ML memiliki perbedaan mendasar dalam cara mereka belajar dan memproses data. Meskipun keduanya menggunakan algoritma untuk mengidentifikasi pola dalam data, mereka melakukannya dengan cara yang berbeda. ML menggunakan metode analisis data yang mengotomatiskan pembuatan model analitik. Ide dasarnya adalah sistem dapat belajar dari data, mengidentifikasi pola, dan membuat keputusan dengan intervensi manusia minimal[20]. ML dapat menggunakan berbagai jenis algoritma dan metode, termasuk *Support Vector Machine* (SVM), *Random Forest* (RF), dan *K-Nearest Neighbor* (KNN), untuk memproses data dan membuat prediksi[21].

Di sisi lain, *Deep Learning* berfokus pada algoritma yang meniru struktur dan fungsi otak manusia, yang dikenal sebagai jaringan saraf tiruan (*Artificial Neural Networks*)[20]. Deep Learning memanfaatkan struktur model jaringan saraf tiruan dengan sejumlah lapisan yang banyak, atau dikenal sebagai *Deep Neural Networks* untuk melakukan pemrosesan data. Dengan memanfaatkan banyaknya lapisan ini, Deep Learning mampu memahami dan mengolah data yang sangat kompleks dan berskala besar dengan lebih efektif[22].

Salah satu perbedaan utama antara DL dan ML adalah dalam cara mereka belajar dari data seperti pada Gambar 2.3. ML biasanya memerlukan intervensi manusia dalam bentuk pemilihan fitur, di mana fitur yang paling relevan dan informatif dipilih untuk digunakan dalam model. Di sisi lain, DL mampu belajar fitur-fitur ini secara otomatis dari data, yang memungkinkan model untuk belajar dan beradaptasi dengan data baru tanpa perlu intervensi manusia[23].



Gambar 2. 3 Perbedaan Machine Learning dan Deep Learning

Sebagai contoh, saat kita ingin membangun model untuk mengenali gambar kucing. Pada teknik ML tradisional, akan dibutuhkan intervensi manusia untuk secara manual merancang fitur yang relevan. Fitur ini bisa mencakup deteksi tepi untuk mengidentifikasi bentuk kucing, analisis warna untuk mencari warna yang biasanya terkait dengan kucing, atau bahkan deteksi tekstur untuk mencari pola bulu kucing. Proses ini memerlukan pengetahuan dan pengalaman yang mendalam dari pengembang model tentang apa yang membuat suatu gambar menjadi gambar kucing dan bagaimana cara terbaik untuk merepresentasikannya dalam bentuk fitur yang dapat diproses oleh model ML.

Sebaliknya, dalam DL, kita cukup memberikan model dengan banyak contoh gambar kucing dan membiarkannya belajar fitur-fitur yang relevan dengan sendirinya. Model DL mungkin akan belajar bahwa bentuk tertentu, warna, dan tekstur berhubungan dengan kucing melalui proses iteratif dan otomatis. Ini dapat mencakup proses yang sama seperti yang dilakukan manusia seperti deteksi tepi atau analisis warna tetapi dilakukan secara otomatis oleh model, tanpa perlu intervensi manusia[24].

Dengan demikian, *Deep Learning* dapat menghemat waktu dan upaya yang diperlukan dalam pemilihan fitur manual dan berpotensi menemukan fitur-fitur

baru yang mungkin tidak terpikirkan oleh manusia. Tetapi, ini tidak berarti bahwa *deep learning* tidak memerlukan supervisi atau intervensi manusia sama sekali.

2.7.1 ANN

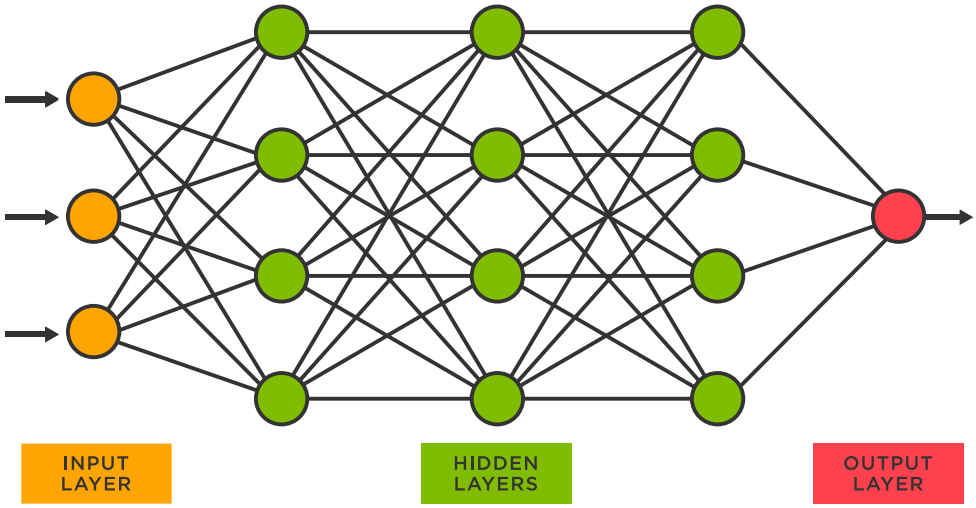
Artificial Neural Networks (ANN) adalah model komputasi yang terinspirasi oleh sistem saraf biologis dan berfungsi untuk meniru cara kerja otak manusia dalam memproses informasi[25]. ANN merupakan elemen kunci dalam *Deep Learning*, yang memungkinkan komputer untuk belajar dari pengalaman dan memahami dunia dalam hal hierarki konsep[26]. Maksud dari hierarki konsep ini adalah ANN dapat memahami dan mempelajari data dalam bentuk yang terstruktur dan berlapis, di mana setiap lapisan menangkap representasi yang berbeda dari data.

Misalnya, penggunaan ANN dalam memprediksi cuaca. Lapisan pertama ANN mungkin belajar untuk mengenali hubungan dasar antara berbagai variabel cuaca seperti suhu, kelembaban, tekanan udara, dan kecepatan angin. Lapisan berikutnya mungkin belajar untuk mengenali pola yang lebih kompleks dalam data, seperti bagaimana kombinasi variabel-variabel ini dapat mengarah ke kondisi cuaca tertentu. Lapisan berikutnya lagi mungkin belajar untuk mengenali pola jangka panjang dan musiman dalam cuaca, seperti bagaimana perubahan suhu dan kelembaban sepanjang tahun dapat mempengaruhi pola cuaca. Dengan kata lain, setiap lapisan membangun dan memperluas pemahaman yang diperoleh dari lapisan sebelumnya, menciptakan sebuah hierarki.

ANN terdiri dari unit-unit pemrosesan yang disebut *neuron* atau sel syaraf. *Neuron* dalam ANN ini diinspirasi oleh *neuron* atau sel syaraf dalam otak manusia. Dalam otak manusia, *neuron* adalah sel-sel saraf yang menerima informasi, memprosesnya, dan mengirimkannya ke *neuron* lain melalui sinapsis (titik kontak antara dua *neuron)*. Dalam ANN, *neuron* adalah unit-unit komputasi yang menerima *input*, melakukan beberapa perhitungan, dan menghasilkan *output*[27].

Struktur ANN biasanya terdiri dari tiga jenis lapisan: *input layers*, *hidden layers*, dan *output layers* seperti pada Gambar 2.4. Lapisan *input* menerima data

masukan dan meneruskannya ke *hidden layers*. Terdapat lebih dari satu *hidden layers* yang melakukan sebagian besar perhitungan dan transformasi data. *Output layers* menerima data dari *hidden layers* dan menghasilkan *input* akhir atau hasil prediksi.



Gambar 2. 4 Struktur ANN

Gambar 2.4 menunjukkan struktur dasar ANN. Pada gambar tersebut, setiap lingkaran mewakili *neuron* dan setiap garis mewakili koneksi antara *neuron*. *Neuron* di *input layers* menerima data masukan dan meneruskannya ke *neuron* di *hidden layers*. *Neuron* di *hidden layers* melakukan perhitungan dan meneruskannya ke *neuron* di *output layers*, yang menghasilkan *input* akhir. ANN bekerja atau dilatih melalui dua tahapan utama, yaitu *forward propagation* dan *backpropagation*.

2.7.1.1 *Forward Propagation*

Forward propagation adalah proses di mana informasi bergerak dari *input layers* ke *output layers*. Pada tahap ini, setiap *neuron* menerima *input*, mengalikan *input* tersebut dengan bobot (*weight)* yang sesuai, menambahkan bias, dan kemudian meneruskannya melalui fungsi aktivasi untuk menghasilkan *output*.

Fungsi aktivasi dalam ANN digunakan untuk mengubah *input neuron* menjadi *output* yang akan diteruskan ke *neuron* berikutnya. Fungsi ini biasanya non-linear dan dapat berbeda-beda tergantung pada jenis ANN dan tujuan aplikasinya. Beberapa fungsi aktivasi yang umum digunakan adalah sigmoid, TANH, ReLU, dan softmax. Fungsi aktivasi memungkinkan ANN untuk memodelkan hubungan yang lebih kompleks dan non-linear antara *input* dan *input*. *Output* yang dihasilkan fungsi aktivasi ini kemudian diteruskan sebagai *input* ke *neuron* di lapisan berikutnya. Persamaan dari proses forward propagation dapat didefinisikan sebagai:

$$y = f(x_1 w_1 + x_2 w_2 + \cdots + x_n w_n + b) \tag{2.1}$$

Di mana:

- y adalah *output* atau prediksi dari jaringan,
- f adalah fungsi aktivasi,
- $x_1, x_2, \ldots, x_n$ adalah *input*,
- $w_1, w_2, \ldots, w_n$ adalah bobot, dan
- b adalah bias

Bobot dan bias adalah parameter penting dalam ANN yang menentukan bagaimana *input* diubah menjadi *output*. Kedua variabel ini akan terus melakukan perubahan atau belajar selama proses pelatihan, karena itulah variable ini memainkan peran penting dalam menentukan kinerja ANN.

Bobot, sering dilambangkan dengan w, adalah nilai yang menentukan sejauh mana *input* mempengaruhi *output neuron*. Setiap *input* ke *neuron* dikalikan dengan bobot yang sesuai sebelum ditambahkan bersama dengan *input* lainnya. Bobot ini dapat dianggap sebagai ukuran kekuatan koneksi antara *neuron*. Jika bobot antara dua *neuron* besar, maka perubahan kecil pada *input neuron* akan menghasilkan perubahan besar pada output *neuron*. Sebaliknya, jika bobotnya kecil, maka perubahan pada *input neuron* akan memiliki sedikit pengaruh pada output *neuron*[28].

Bias, sering dilambangkan dengan b, adalah nilai tambahan yang ditambahkan ke output *neuron* setelah semua *input* telah dikalikan dengan bobot mereka. Bias memungkinkan *neuron* untuk menghasilkan output yang berbeda meskipun semua *input*nya nol. Dengan kata lain, bias memungkinkan *neuron* untuk bergeser outputnya ke atas atau ke bawah, yang bisa sangat penting untuk memodelkan data dengan benar[29].

Dalam ANN, bobot dan bias biasanya diinisialisasi secara acak dan kemudian diperbarui secara iteratif melalui proses pelatihan dengan tujuan meminimalkan *loss function*[30]. Proses ini melibatkan penggunaan algoritma seperti *gradient descent*, yang secara efektif mencoba menemukan *set* bobot dan bias yang menghasilkan eror terkecil antara prediksi ANN dan nilai target yang sebenarnya.

2.7.1.2 *Backpropagation*

Setelah hasil prediksi didapatkan, proses pelatihan dilanjutkan ke tahap *backpropagation. Backpropagation* adalah proses di mana ANN memperbarui bobotnya berdasarkan kesalahan yang dihasilkan selama *forward propagation*. Ini dimulai dengan menghitung *error* antara output yang dihasilkan oleh ANN dan nilai target (sebenarnya). Proses menghitung *error* ini biasa disebut *loss function*. Dengan kata lain, *loss function* memberikan ukuran seberapa baik atau buruk model dalam memprediksi output yang diharapkan[19].

Ada berbagai jenis *loss function* yang dapat digunakan, tergantung pada jenis masalah yang hadapi. Seperti contoh *mean squared error* (MSE), merupakan *loss function* yang paling umum untuk masalah regresi[31]. MSE menghitung rata-rata dari kuadrat perbedaan antara nilai prediksi dan nilai sebenarnya. Persamaan dari MSE dapat didefinisikan sebagai:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{2.2}$$

Di mana:

- n adalah jumlah sampel
- y_i adalah nilai sebenarnya
- $\hat{y}_i$ adalah nilai prediksi

Setelah menghitung *loss function*, langkah selanjutnya dalam proses pelatihan ANN adalah menghitung gradien dari *loss function* terhadap bobot dan bias. Gradien ini memberikan informasi tentang seberapa cepat *loss function* berubah terhadap perubahan bobot dan bias. Dengan kata lain, gradien ini memberikan arah di mana kita harus mengubah bobot dan bias untuk mencapai nilai minimum dari *loss function*[32]. Untuk menghitung gradien dari *loss function* terhadap bobot dan bias, perlu dihitung turunan parsial dari *loss function* ini terhadap bobot dan bias. Seperti contoh persamaan dari menghitung gradien dari fungsi MSE, dapat didefinisikan sebagai:

$$\frac{\partial MSE}{\partial w} = \frac{-2}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)\,.\,x_i \tag{2.3}$$

Setelah gradien dihitung, bobot dan bias dapat diperbarui (*update)* menggunakan aturan *gradient descent*, yang dapat didefinisikan sebagai:

$$w = w - \alpha\frac{\partial \mathrm{MSE}}{\partial w} \tag{2.4}$$

$$b = b - \alpha\frac{\partial \mathrm{MSE}}{\partial b} \tag{2.5}$$

Di mana:

- α adalah *learning rate*

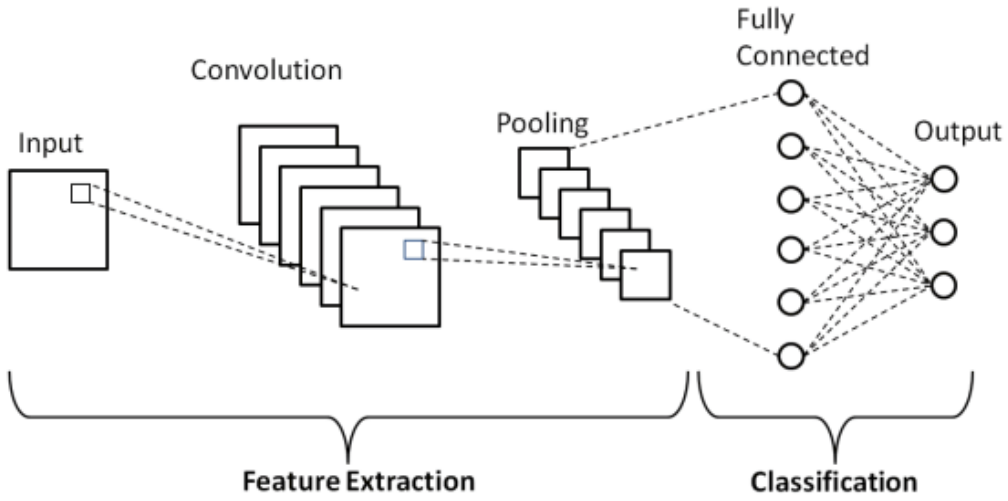
Learning rate adalah parameter penting dalam proses pelatihan ANN dan algoritma pelatihan mesin lainnya. *Learning rate*, sering dilambangkan dengan α, parameter ini mengontrol seberapa besar langkah yang kita ambil saat melakukan

update bobot dan bias dalam proses gradient descent. Dengan kata lain, *learning rate* menentukan seberapa cepat atau lambat model kita belajar dari data[33].

Menentukan *learning rate* yang tepat sangat penting untuk mencapai konvergensi dalam proses pelatihan. Jika *learning rate* terlalu tinggi, model mungkin akan melompat melewati *global minima* (nilai terkecil *loss function)* dan gagal konvergen. Sebaliknya, jika *learning rate* terlalu rendah, algoritma mungkin akan membutuhkan waktu yang sangat lama untuk konvergen atau bahkan bisa terjebak di *local minima* dan tidak mencapai *global minima*[34].

2.7.2 CNN

Convolutional Neural Networks (CNN) adalah varian khusus dari *Artificial Neural Networks* (ANN) yang dirancang untuk mengolah data dengan struktur *grid*, seperti pada *input* gambar. Pada Gambar 2.5, memiliki struktur yang mirip dengan ANN biasa Gambar 2.4, tetapi dengan penambahan lapisan konvolusional dan *pooling* yang diikuti oleh lapisan *fully connected* ini memungkinkan mereka untuk memproses data berstruktur *grid* dengan lebih efisien. CNN mengambil inspirasi dari sistem visual biologis, khususnya dari cara otak manusia memproses informasi visual. Dengan kemampuan unik ini, CNN telah menjadi algoritma kunci dalam bidang *computer vision*, yang memungkinkan komputer untuk melihat dan memahami gambar dengan cara yang mirip dengan manusia[35].



Gambar 2. 5 Struktur CNN

Setiap lapisan konvolusional menggunakan *filter* untuk memindai *input* dan menghasilkan *feature map*, dan setiap lapisan *pooling* mengurangi dimensi dari *feature map* tersebut. Lapisan konvolusional dan *pooling* ini memungkinkan CNN untuk belajar secara efisien tentang fitur-fitur lokal dalam citra, seperti tepi, tekstur, dan pola. Fitur-fitur lokal ini kemudian dapat digabungkan untuk memahami fitur yang lebih kompleks dan abstrak. Lapisan *fully connected* kemudian mengambil fitur-fitur ini dan menggunakannya untuk mengklasifikasikan gambar[36].

Adapun itu, CNN sering dikaitkan dengan pengolahan citra atau gambar karena arsitektur mereka sangat efektif dalam menangani data yang memiliki struktur *grid* spasial, seperti piksel dalam gambar. Konvolusi, yang merupakan operasi inti dalam CNN, sangat cocok untuk mendeteksi pola lokal dalam gambar, seperti tepi, sudut, dan tekstur. Namun, penggunaan CNN tidak harus selalu berhubungan dengan citra atau gambar. Meskipun metode ini paling terkenal dalam aplikasi pengenalan gambar, CNN juga dapat digunakan dalam berbagai jenis data lain yang memiliki struktur *grid* spasial, seperti:

1. **Analisis data spasial**: CNN dapat digunakan untuk menganalisis dan memprediksi data geospasial, seperti peta cuaca atau data topografi.

2. **Pengolahan Suara**: CNN dapat digunakan dalam pengenalan suara atau analisis sinyal audio, di mana data audio dapat dianggap sebagai sinyal 1 dimensi dengan struktur *grid*.
3. **Analisis teks**: meskipun kurang umum digunakan, namun CNN juga dapat digunakan dalam pengolahan bahasa alami untuk menganalisis teks. Konvolusi 1D dapat digunakan untuk menangkap pola lokal dalam kalimat atau dokumen.

Jadi, meskipun CNN paling sering digunakan untuk pengolahan citra, arsitektur ini cukup fleksibel untuk digunakan dalam berbagai aplikasi lain yang melibatkan data dengan struktur *grid* spasial.

2.7.2.1 *Convolutional Layers*

Dalam Gambar 2.5 lapisan *convolution* atau konvolusional digambarkan dengan beberapa lapisan (layer). Ini karena lapisan konvolusional sebenarnya terdiri dari beberapa "*feature maps*", yang masing-masing *feature maps* merupakan hasil dari operasi konvolusi menggunakan *filter* atau *kernel* yang berbeda. Operasi konvolusi adalah operasi matematika yang menggabungkan *input* dan *filter* untuk menghasilkan satu set *feature map*[37].

Setiap *filter* dalam lapisan konvolusional adalah matriks kecil dari bobot. Ukuran matriks ini biasanya kecil, seperti 3x3 atau 5x5, dan setiap elemen dalam matriks ini adalah bobot yang dapat dipelajari. Misalkan kita memiliki *filter* 3x3, maka matriks bobotnya terlihat seperti ini:

$$\begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \end{bmatrix}$$

Di mana w_{ij} adalah bobot pada baris ke-i dan kolom ke-j. Saat melakukan operasi konvolusi, *filter* ini digeser dari kiri ke kanan gambar *input*. Kemudian *filter* melakukan perkalian elemen-per-elemen antara matriks bobot dan bagian gambar *input*, dan kemudian menjumlahkan hasilnya untuk mendapatkan satu nilai. Nilai ini kemudian menjadi elemen dalam *feature map*. Bobot dalam *filter* ini dipelajari

selama proses pelatihan. Tujuannya adalah untuk menyesuaikan bobot ini sehingga *filter* dapat mendeteksi fitur tertentu dalam gambar, seperti tepi, sudut, atau tekstur. Misalnya, *filter* yang telah dilatih untuk mendeteksi tepi vertikal mungkin memiliki bobot yang tinggi di kolom tengah dan bobot yang rendah di kolom lainnya.

2.7.2.2 *Pooling Layers*

Setelah lapisan konvolusional, biasanya diikuti oleh lapisan *pooling*. Tujuan dari lapisan *pooling* adalah untuk mengurangi dimensi spasial dari *feature map* yang dihasilkan oleh lapisan konvolusional sebelumnya, sambil mempertahankan informasi penting. Dengan mengurangi dimensi ini, lapisan *pooling* membantu mengurangi jumlah parameter dan komputasi dalam jaringan[38].

Lapisan *pooling* bekerja dengan mengambil jendela kecil atau *sub-region* (misalnya, 2x2 piksel) dari *feature map* dan menerapkan operasi *pooling* ke nilai-nilai dalam jendela tersebut. Ada beberapa jenis operasi *pooling* yang dapat digunakan, tetapi dua yang paling umum adalah *max pooling* dan *average pooling*. *Max pooling* mengambil nilai maksimum dari jendela tersebut, sementara *average pooling* mengambil rata-rata dari nilai-nilai tersebut. Misalnya, jika kita memiliki jendela 2x2 dari *feature map* seperti ini:

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

Maka operasi *max pooling* akan menghasilkan nilai 4 (nilai maksimum dari jendela tersebut), sementara operasi *average pooling* akan menghasilkan nilai 2.5 (rata-rata dari nilai-nilai tersebut). Setelah operasi *pooling* diterapkan, jendela tersebut kemudian digeser (biasanya dengan *stride* 2 atau 2 piksel) ke sepanjang *feature map*, dan operasi *pooling* diterapkan lagi. Proses ini berlanjut sampai seluruh *feature map* telah diproses, menghasilkan *feature map* yang lebih kecil.

Dengan mengurangi dimensi *feature map*, lapisan *pooling* membantu membuat model lebih invarian terhadap pergeseran dan rotasi dari gambar *input*. Dengan kata lain, bahkan jika gambar digeser atau diputar sedikit, output dari

lapisan *pooling* akan tetap sama. Ini sangat penting dalam banyak aplikasi pengenalan gambar, di mana kita ingin model kita mampu mengenali objek dalam gambar, tidak peduli bagaimana objek tersebut diposisikan.

2.7.2.3 *Fully Connected Layers*

Setelah lapisan konvolusional dan *pooling*, CNN biasanya memiliki satu atau lebih lapisan yang disebut lapisan *fully connected*. Lapisan ini disebut *fully connected* karena setiap *neuron* dalam lapisan ini terhubung ke setiap *neuron* di lapisan sebelumnya. Ini mirip dengan bagaimana *neuron* terhubung dalam jaringan saraf biasa atau ANN[39].

Lapisan *fully connected* berfungsi untuk mengambil semua *output* dari lapisan sebelumnya (yang mungkin merupakan *feature map* dari lapisan konvolusional atau *pooling*) dan mengubahnya menjadi satu vektor 1D[40]. Vektor ini kemudian dapat diproses oleh lapisan *fully connected*, yang biasanya menggunakan fungsi aktivasi seperti ReLU atau *sigmoid*.

Tujuan dari lapisan *fully connected* adalah untuk melakukan klasifikasi pada fitur yang telah diekstraksi oleh lapisan sebelumnya. Misalnya, jika model CNN digunakan untuk mengenali gambar kucing dan anjing, lapisan *fully connected* akan mengambil fitur yang telah diekstraksi oleh lapisan konvolusional dan *pooling* (seperti tepi, tekstur, dll.) dan menggunakannya untuk memutuskan apakah gambar tersebut adalah kucing atau anjing.

Output dari lapisan *fully connected* terakhir (juga dikenal sebagai lapisan *output*) kemudian digunakan untuk membuat prediksi. Jika dalam kasus klasifikasi gambar, lapisan *output* mungkin memiliki jumlah *neuron* yang sama dengan jumlah kelas yang ada (misalnya, satu *neuron* untuk kucing dan satu *neuron* untuk anjing), dan *neuron* dengan aktivasi tertinggi akan menentukan kelas prediksi.

2.7.2.4 *Activation Function*

Activation function atau fungsi aktivasi adalah komponen penting dalam arsitektur CNN dan ANN pada umumnya. Fungsi ini diterapkan pada setiap *neuron* dalam jaringan dan bertindak sebagai mekanisme pengambilan keputusan, menentukan apakah *neuron* tersebut harus "diaktifkan" atau tidak berdasarkan *input* yang diterimanya[41].

Secara teknis, fungsi aktivasi mengambil *input* (hasil dari penjumlahan bobot *input* dan bias) dan mengubahnya menjadi *output* yang akan digunakan sebagai *input* untuk *neuron* di lapisan berikutnya dalam jaringan. *Output* ini biasanya berada dalam rentang tertentu, yang ditentukan oleh jenis fungsi aktivasi yang digunakan.

Fungsi aktivasi memainkan peran penting dalam memperkenalkan non-linearitas ke dalam model. Tanpa fungsi aktivasi, setiap lapisan dalam *neural network* hanya akan melakukan operasi linier pada *input* (seperti penjumlahan atau perkalian), di seluruh jaringan, tidak peduli seberapa banyak lapisan yang dimilikinya, akan setara dengan operasi linier tunggal. Ini berarti jaringan tersebut tidak akan dapat mempelajari pola yang lebih kompleks dari data, maka hal ini berlawanan dari tujuan ANN.

Dalam implementasi jaringan saraf, terdapat berbagai jenis fungsi aktivasi yang dapat digunakan, dan masing-masing memiliki karakteristik serta penggunaannya masing-masing. Pemilihan fungsi aktivasi yang tepat sangat penting karena dapat mempengaruhi kinerja dan kecepatan pelatihan jaringan. Seperti contoh fungsi aktivasi ReLU (*Rectified Linear Unit*) yang sangat populer dalam CNN karena efisiensi komputasionalnya. ReLU mengubah semua nilai *input* negatif menjadi nol dan membiarkan semua nilai positif tetap tidak berubah. Meskipun sederhana, ReLU telah terbukti efektif dalam berbagai tugas dan merupakan pilihan *default* untuk banyak arsitektur CNN. Fungsi ini didefinisikan sebagai:

$$f(x) = \max(0, x) \tag{2.6}$$

Di mana:

- x adalah *input*

Yang berarti, jika *input* x lebih besar dari 0, fungsi akan menghasilkan x sebagai *output*. Jika x kurang dari atau sama dengan 0, fungsi akan menghasilkan 0.

2.7.3 Permasalahan Dalam *Deep Learning*

Meskipun *deep dearning* (DL) telah mencapai kemajuan yang signifikan dalam beberapa tahun terakhir dan telah berhasil diterapkan dalam berbagai bidang, namun masih ada sejumlah tantangan dan permasalahan yang harus dihadapi. Sebagai teknologi yang relatif baru dan berkembang pesat, DL masih dalam tahap eksplorasi dan penemuan, dan ada banyak aspek yang masih perlu dipahami dan ditingkatkan.

Salah satu tantangan utama dalam DL adalah bagaimana memilih dan menyesuaikan model yang tepat untuk suatu tugas atau permasalahan. Meski model DL telah menunjukkan performa yang luar biasa dalam berbagai tugas, namun tidak ada satu model pun yang cocok untuk semua jenis tugas. Oleh karena itu, pemilihan dan penyesuaian model yang tepat untuk suatu tugas tertentu bisa menjadi tantangan yang cukup besar[41].

Selain itu, DL juga menghadapi tantangan dalam hal interpretasi dan transparansi. Model DL sering kali dianggap sebagai *black box*, di mana sulit untuk memahami bagaimana model tersebut membuat prediksi. Hal ini bisa menjadi permasalahan dalam aplikasi di mana transparansi dan interpretasi penting, seperti dalam bidang kesehatan atau hukum.

Model DL yang memiliki arsitektur yang kompleks dan banyak parameter, memerlukan banyak sumber daya komputasi untuk melatih dan menjalankan

model. Ini mencakup CPU atau GPU yang kuat, serta memori yang cukup untuk menyimpan model dan data yang digunakan. Pelatihan model seperti ini bisa memakan waktu yang sangat lama dan membutuhkan banyak komputasi, yang bisa menjadi masalah jika sumber daya komputasi terbatas. Selain itu, model yang besar dan kompleks juga bisa sulit untuk di-*deploy* atau digunakan dalam aplikasi *real-time* atau pada perangkat dengan sumber daya terbatas, seperti *smartphone* atau IoT *devices*.

Namun, dua permasalahan yang paling umum dan sering dihadapi dalam DL adalah *overfitting* dan *underfitting*. *Overfitting* terjadi ketika model terlalu kompleks dan "belajar terlalu baik" dari data pelatihan, sehingga performanya buruk pada data yang belum pernah dilihat sebelumnya. Sebaliknya, *underfitting* terjadi ketika model terlalu sederhana dan tidak mampu belajar dari data pelatihan.

2.7.3.1 *Overfitting*

Overfitting adalah fenomena dalam *deep learning* (DL) di mana model yang telah dilatih menunjukkan performa yang sangat baik pada data pelatihan, tetapi performanya menurun secara signifikan saat diuji pada data yang belum pernah dilihat sebelumnya. Ini biasanya terjadi ketika model terlalu kompleks dan memiliki terlalu banyak parameter. Parameter dalam DL ini merujuk pada variabel internal model yang dipelajari selama proses pelatihan dan digunakan untuk membuat prediksi. Jumlah parameter dalam model sering kali mencerminkan kompleksitas model. Maka dari itu, jika model memiliki terlalu banyak parameter, ia akan menghafal data pelatihan dengan sangat baik, termasuk *noise* atau variasi acak dalam data, yang seharusnya diabaikan.

Dengan kata lain, *overfitting* terjadi ketika model belajar terlalu detail atau terlalu spesifik pada data pelatihan sehingga gagal untuk menggeneralisasi pola yang ada pada data yang belum pernah dilihat sebelumnya.

Overfitting dalam model *deep learning* dapat disebabkan oleh berbagai faktor. Berikut adalah beberapa penyebab utama dari *overfitting*:

1. **Kompleksitas Model**: Salah satu penyebab utama *overfitting* adalah model yang terlalu kompleks. Model dengan banyak parameter atau lapisan cenderung memiliki kapasitas yang tinggi, yang berarti mereka mampu mempelajari pola yang sangat kompleks dan spesifik dari data pelatihan. Oleh karena itu, penting untuk memilih arsitektur model yang yang sesuai dengan kompleksitas data dan tugas yang dihadapi.
2. **Kurangnya Data Pelatihan**: *Overfitting* juga dapat disebabkan oleh kurangnya data pelatihan. Jika jumlah data pelatihan yang tersedia relatif kecil dibandingkan dengan kompleksitas model, model mungkin akan "menghafal" data pelatihan daripada belajar pola umum yang ada dalam data. Oleh karena itu, disarankan mengumpulkan banyak data yang bervariasi.
3. **Pelatihan Model untuk Durasi yang Terlalu Lama**: Jika model dilatih untuk jumlah *epoch* atau iterasi yang terlalu banyak, model mungkin akan mulai menghafal data pelatihan dan mengakibatkan *overfitting*.

Terdapat beberapa cara yang dapat digunakan untuk mendeteksi *overfitting*. Salah satu cara yang paling mudah adalah dengan membagi *dataset* menjadi 3 yaitu: data latih, validasi, dan tes. Selain itu, kita juga dapat memantau kurva belajar model selama proses pelatihan. Kurva belajar adalah plot yang menunjukkan metrik evaluasi seperti mAP, *recall*, dan *precision* dari model seiring berjalannya waktu atau iterasi. Metrik-metrik ini dihitung baik untuk data latih dan validasi yang kemudian digambarkan dalam bentuk grafik. Dengan melihat kurva belajar, kita dapat memahami bagaimana model belajar seiring waktu dan apakah model tersebut mengalami *overfitting*. Jika model menunjukkan performa yang sangat baik pada metrik evaluasi untuk data latih tetapi performanya menurun secara signifikan pada data validasi, ini bisa menjadi indikasi bahwa model mengalami *overfitting*.

2.7.3.2 *Underfitting*

Underfitting dalam *deep learning* terjadi ketika model tidak mampu mempelajari pola yang ada dalam data pelatihan dengan cukup baik. Maksudnya, model tidak mampu menangkap atau mempelajari pola atau struktur yang ada dalam data, yang mengakibatkan performa yang buruk tidak hanya pada data pelatihan, tetapi juga pada data yang belum pernah dilihat sebelumnya. Maka dari itu *underfitting* adalah kebalikannya dari o*verfitting*. Jika *overfitting* adalah kondisi di mana model belajar terlalu baik dari data pelatihan dan gagal untuk menggeneralisasi pola yang ada pada data yang belum pernah dilihat sebelumnya, *underfitting* adalah kondisi di mana model belajar terlalu sedikit dari data latih dan gagal untuk menangkap pola yang ada dalam data. Model yang mengalami *underfitting* biasanya memiliki performa atau hasil evaluasi yang buruk pada data latih dan validasi, ini menandakan bahwa model tidak mampu mempelajari pola yang ada dalam data dengan cukup baik.

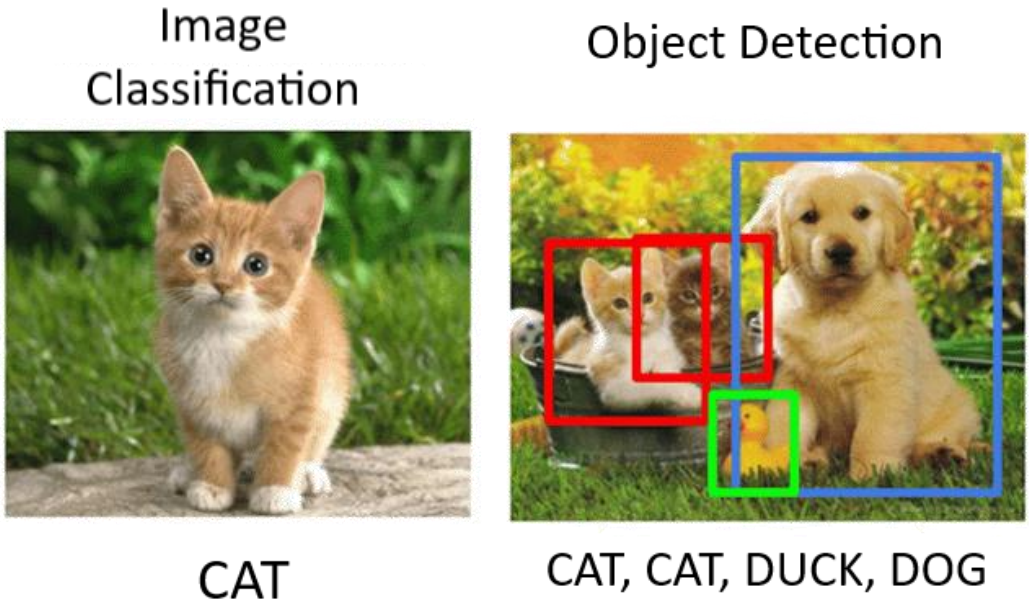
Salah satu penyebab utama terjadinya *underfitting* adalah struktur model yang terlalu sederhana. Struktur model yang terlalu sederhana dengan sedikit parameter atau lapisan mungkin tidak memiliki kapasitas yang cukup untuk mempelajari pola yang kompleks dalam data. Selain itu, jika data pelatihan tidak mencakup variasi yang cukup dalam pola yang ada dalam data, model mungkin tidak akan mampu mempelajari pola data latih dengan cukup baik.

Sama seperti *overfitting*, untuk mendeteksi *underfitting* salah satunya dengan cara memantau performa model selama proses pelatihan. Jika model menunjukkan performa yang buruk pada data pelatihan dan data validasi pada iterasi tertentu, ini bisa menjadi indikasi bahwa model yang dilatih mengalami *underfitting*.

2.8 *Object Detection*

Object detection merupakan salah satu cabang pada bagian *computer vision* yang berfokus pada identifikasi serta penentuan lokasi objek spesifik pada sebuah

citra digital. Objek yang diidentifikasi dapat berupa apa saja, mulai dari manusia, kendaraan, hingga hewan. Prinsip kerja dari *object detection* melibatkan dua proses utama, yaitu klasifikasi dan lokalisasi. Klasifikasi bertujuan untuk mengidentifikasi kelas atau jenis objek yang ada, sementara lokalisasi berfungsi untuk menentukan posisi atau koordinat objek dalam gambar, seperti pada gambar 2.6. Dengan menggabungkan kedua proses ini, *object detection* mampu memberikan informasi lengkap tentang apa yang ada didalam gambar dan di mana objek tersebut berada.



Gambar 2. 6 Perbedaan Klasifikasi Gambar dengan Object Detection

Pada bidang *computer vision*, selain *object detection*, terdapat juga konsep yang dikenal sebagai *image classification*. Konsep ini berfokus pada pengidentifikasian kategori atau kelas utama dari seluruh gambar, tanpa menentukan lokasi atau jumlah objek spesifik dalam gambar tersebut.

Sebagai contoh pada gambar 2.6, *image classification* dapat mengidentifikasi gambar sebagai "kucing" atau "anjing," tetapi tidak menentukan di mana kucing atau anjing tersebut berada dalam gambar, atau berapa banyak kucing atau anjing yang ada. Sementara itu, *object detection* tidak hanya mengidentifikasi jenis objek yang ada dalam gambar, tetapi juga menentukan lokasi

dan jumlah objek tersebut. Dengan kata lain, *object detection* memberikan informasi yang lebih detail dan spesifik tentang gambar, termasuk kelas, lokasi, dan jumlah objek. Perbedaan ini menjadikan *object detection* bisa diimplementasikan ke berbagai aplikasi yang rumit, seperti kamera pengawas hingga *self driving car*.

Berbagai teknik object detection telah dikembangkan untuk meningkatkan efisiensi dan akurasi dalam mengidentifikasi objek pada citra digital. Salah satu teknik awal yang diperkenalkan adalah *sliding windows*. Di mana jendela berukuran tertentu digeser melintasi citra untuk mencari objek. Meskipun sederhana, pendekatan ini memerlukan banyak komputasi karena harus memeriksa setiap kemungkinan lokasi dan ukuran jendela.

Selanjutnya, untuk meningkatkan efisiensi, diperkenalkanlah *image pyramids*. Teknik ini melibatkan penciptaan beberapa versi skala atau resolusi dari citra asli. Dengan teknik ini dapat membagi tugas deteksi objek dalam berbagai ukuran dengan lebih efisien. Sehingga, objek yang lebih besar dapat dideteksi pada skala yang lebih rendah, sementara objek yang lebih kecil pada skala yang lebih tinggi.

Teknik lainnya yaitu *region proposals*, teknik ini memeriksa setiap bagian dari citra. Teknik ini bekerja dengan cara mengidentifikasi wilayah-wilayah yang berpotensi mengandung objek. Kemudian, wilayah yang mengandung objek akan dianalisis lebih lanjut untuk mengurangi beban komputasi dan meningkatkan kecepatan deteksi. Secara keseluruhan, teknik-teknik dalam *object detection* terus berkembang seiring dengan kemajuan teknologi dan kebutuhan aplikasi yang spesifik. Setiap teknik memiliki kelebihan dan kekurangannya masing-masing, dan pemilihan teknik yang tepat bergantung pada permasalahan atau tujuan aplikasi yang diinginkan.

Setelah memahami berbagai teknik *object detection*, penting untuk diketahui bahwa perkembangan terbaru dalam bidang ini sering kali melibatkan penggunaan *deep learning*, khususnya *Convolutional Neural Networks* (CNN). Karena CNN adalah jenis jaringan saraf yang sangat efektif dalam mengenali pola

pada citra digital. Oleh karena itu CNN sering digunakan untuk pembuatan aplikasi *object detection*.

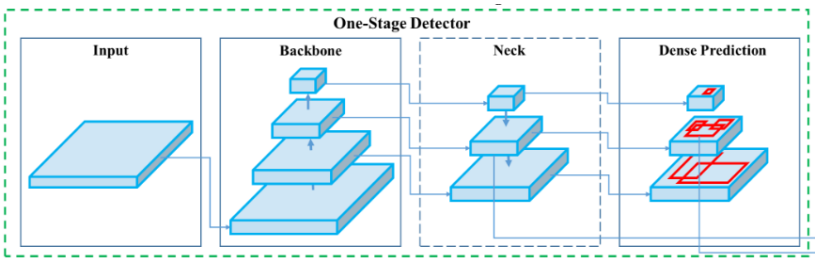
2.8.1 YOLO

You only look once, atau biasa disebut YOLO, merupakan sebuah model *object detection* dalam *computer vision*. Berbeda dengan model *object detection* lainnya yang mungkin memerlukan beberapa tahapan analisis dan evaluasi. YOLO mengambil pendekatan yang unik tetapi efisien dengan melakukan deteksi dan klasifikasi objek dalam satu kali evaluasi terhadap citra *input*. Selain kecepatan deteksi, YOLO juga bisa mempertahankan akurasi prediksi yang tinggi.

Dengan arsitektur yang teroptimasi, YOLOv4 (versi keempat) mampu mencapai hingga 65 *frame per second* (FPS) dengan menggunakan GPU Tesla V100[42]. Ini membuat YOLO menjadi salah satu algoritma deteksi objek yang tercepat pada saat ini. Pengembangan YOLO telah membuka peluang baru dalam aplikasi *real-time* dan pengembangan teknologi deteksi objek.

2.8.1.1 Arsitektur YOLOv4

YOLOv4, sebagai salah satu versi dalam seri YOLO, menghadirkan peningkatan signifikan dalam hal efisiensi dan akurasi. Meskipun telah ada versi lebih baru seperti YOLOv8, YOLOv4 tetap menjadi salah satu implementasi yang penting dan banyak digunakan dalam deteksi objek. Gambar 2.7 menunjukkan arsitektur YOLOv4, yang mencakup bagian *input*, *backbone*, *neck*, dan *dense prediction*.



Gambar 2. 7 Arsitektur YOLOv4

Pada tahap awal atau bagian *input* arsitektur YOLOv4, *input* citra dirubah menjadi ukuran 608×608 piksel untuk yang versi standar. Proses perubahan ukuran ini dilakukan dengan menggunakan teknik interpolasi *bilinear*. Teknik ini mengestimasi nilai piksel baru berdasarkan empat piksel terdekat dari posisi yang sedang diestimasi. Dengan menggunakan pendekatan ini, YOLOv4 dapat mengubah ukuran *input* citra menjadi dimensi yang seragam tanpa kehilangan informasi penting, sekaligus menjaga efisiensi komputasi.

Citra dengan resolusi yang sudah ditentukan kemudian dinormalisasi, sehingga nilai pikselnya berada dalam rentang 0 hingga 1. Normalisasi ini penting untuk memastikan bahwa model dapat bekerja dengan efisien dan mengurangi potensi masalah numerik saat pelatihan. Selain itu, pada tahap ini juga melakukan *augmentation* data diterapkan untuk meningkatkan ketahanan model terhadap variasi dalam data baru. *Augmentation* mungkin termasuk rotasi, pemotongan, dan perubahan skala. Setelah semua tahapan ini, citra yang telah diolah siap untuk diteruskan ke bagian *Backbone* untuk ekstraksi fitur.

Bagian *backbone* dari arsitektur YOLOv4 bertanggung jawab untuk ekstraksi fitur dari citra yang telah diolah pada bagian *input*. *Backbone* YOLOv4 terdiri dari beberapa lapisan konvolusional yang saling terhubung, yang biasanya diatur dalam bentuk hierarki, seperti pada Gambar 2.7. Setiap lapisan dalam hierarki ini bertanggung jawab untuk mengenali fitur pada tingkat kompleksitas yang berbeda. Lapisan awal biasanya mengenali fitur sederhana seperti garis dan tepi, sementara lapisan yang lebih dalam dapat mengenali struktur yang lebih kompleks dan abstrak.

YOLOv4 menggunakan arsitektur CSPDarknet53 sebagai *backbone*. CSPDarknet53 menggabungkan teknik *cross stage hierarchical* (CSH) dengan Darknet53, yang memungkinkan ekstraksi fitur yang lebih efisien dan efektif. Teknik CSH membagi fitur menjadi dua bagian pada setiap tahap dan menggabungkannya kembali pada tahap berikutnya, memungkinkan gradien *loss*

function untuk mengalir lebih bebas melalui jaringan[42]. Dengan menggunakan teknik-teknik tadi maka kinerja pelatihan dan akurasi model akan meningkat.

Setelah proses ekstraksi fitur dalam bagian *backbone*, fitur-fitur ini kemudian diproses melalui serangkaian fungsi aktivasi. Dalam YOLOv4, fungsi aktivasi seperti *leaky* ReLU dan *mish* digunakan. *Leaky* ReLU digunakan untuk mengatasi masalah *vanishing gradient*, sementara *mish*, yang merupakan fungsi aktivasi yang lebih baru, telah terbukti memberikan kinerja yang lebih baik dalam beberapa kasus.

Kemudian terdapat bagian *neck* dari arsitektur YOLOv4 bertugas sebagai penghubung antara bagian *backbone* dan *dense prediction*. Bagian ini berfungsi untuk memadukan fitur-fitur yang diekstrak oleh *backbone*. Selain itu alur kerja pada bagian *neck* juga dikenal sebagai *feature pyramid network* (FPN) yang bertujuan untuk menggabungkan fitur beresolusi tinggi dengan fitur semantik yang kaya dari berbagai skala dalam sebuah piramida fitur[43].

Dalam implementasinya YOLOv4 tidak menggunakan FPN secara tradisional. Akan tetapi, YOLOv4 menggunakan kombinasi dari *path aggregation network* (PANet) dan *spatial pyramid pooling* (SPP) untuk menggabungkan fitur dari berbagai skala gambar[42]. PANet memungkinkan informasi untuk mengalir ke atas dan ke bawah piramida fitur (Gambar 2.7). Ini untuk memastikan bahwa semua *level* atau lapisan memiliki informasi dari *level* lain. Sementara itu, SPP digunakan untuk mengekstrak fitur pada skala yang berbeda dan menggabungkannya, yang memungkinkan deteksi objek pada berbagai ukuran. Pada akhir bagian *neck*, fitur-fitur ini kemudian diteruskan ke bagian *dense prediction* untuk proses deteksi objek.

Dalam bagian *dense prediction*, fitur-fitur yang diterima dari bagian *neck* diubah menjadi serangkaian *bounding box*, masing-masing dengan skor probabilitas untuk setiap kelas objek yang mungkin. *Bounding box* ini merepresentasikan lokasi potensial objek dalam citra, sementara skor probabilitas

menunjukkan keyakinan model bahwa objek tertentu ada di dalam *bounding box* tersebut.

YOLOv4 pada bagian *dense prediction* menggunakan tiga skala berbeda untuk prediksi, yaitu skala 13×13, 26×26, dan 52×52 piksel. Setiap skala ini dirancang untuk menangani objek dengan ukuran yang berbeda. Skala 13×13 misalnya digunakan untuk mendeteksi objek berukuran besar, skala 26×26 digunakan untuk mendeteksi objek berukuran menengah, dan skala 52×52 digunakan untuk mendeteksi objek berukuran kecil. Setiap sel dalam grid ini bertanggung jawab untuk memprediksi sejumlah *bounding box* dan skor kelas. Dengan demikian, YOLOv4 dapat mendeteksi objek pada berbagai ukuran dengan efektif. Dan pada akhirnya setiap *bounding box* yang dihasilkan oleh *dense prediction* akan menghasilkan empat nilai yang merepresentasikan koordinat x dan y pada gambar sebagai pusat *bounding box* dan lebar (w) serta tingginya (h).

Setelah *bounding box* dan skor kelas diprediksi, selama proses pelatihan, model kemudian menghitung *loss function* antara prediksi dan *ground truth*. YOLOv4 disini menggunakan kombinasi dari beberapa *loss function*. Untuk koordinat bounding box dan ukuran, YOLOv4 menggunakan *mean square error* (MSE). MSE mengukur perbedaan kuadrat antara nilai prediksi dan nilai sebenarnya, memberikan penalti yang lebih besar untuk kesalahan yang lebih besar. Untuk kelas objek, YOLOv4 menggunakan *cross-entropy* (CE). CE mengukur perbedaan antara distribusi probabilitas yang diprediksi oleh model dan distribusi probabilitas sebenarnya. Selain itu, YOLOv4 juga menggunakan IoU *loss* untuk mengukur perbedaan antara IoU dari prediksi dan *ground truth*. Dengan menggunakan kombinasi *loss function* ini, YOLOv4 dapat belajar untuk memprediksi *bounding box* dan kelas objek dengan lebih akurat.

Setelah bagian *dense prediction*, proses dalam arsitektur YOLOv4 memang telah selesai. Namun, dalam implementasinya, ada beberapa langkah *post-processing* yang dilakukan untuk memastikan hasil deteksi objek yang optimal. Langkah-langkah ini tidak secara eksplisit digambarkan dalam diagram arsitektur,

tetapi sangat penting dalam operasi model. Salah satu langkah atau teknik yang digunakan dalam *post-processing* yaitu *non maximum suppression* (NMS). NMS adalah teknik yang digunakan untuk mengatasi masalah *bounding box* yang tumpang tindih. Dalam permasalahan *object detection*, sering kali model akan memprediksi beberapa *bounding box* yang tumpang tindih pada objek yang sama. NMS bekerja dengan mempertahankan *bounding box* dengan skor probabilitas tertinggi dan menghapus *bounding box* lainnya yang memiliki *Intersection over Union* (IoU) yang tinggi dengan bounding box tersebut.

2.8.2 *Dataset*

Dataset merupakan elemen yang penting dalam bidang ML dan DL. Secara umum, *dataset* dapat didefinisikan sebagai kumpulan data yang digunakan untuk melatih dan menguji model ML atau DL. *Dataset* biasanya terdiri dari dua bagian utama, yaitu data fitur dan label. Data fitur adalah variabel independen yang digunakan sebagai *input* oleh model, sementara label adalah variabel dependen yang menjadi target prediksi model.

Untuk memastikan bahwa model dapat menggeneralisasi dengan baik ke data baru, *dataset* biasanya dibagi menjadi tiga bagian, yaitu *training set*, *validation set*, dan *test set*. *Training set* digunakan untuk melatih model, *validation* set digunakan untuk menyetel *hyperparameter* dan mencegah *overfitting*, dan *test set* digunakan untuk mengevaluasi kinerja akhir model. Pembagian ini memungkinkan kita untuk mengukur sejauh mana model mampu memprediksi data baru yang tidak pernah dilihatnya sebelumnya.

Pada metode *object detection*, seperti yang diimplementasikan dalam YOLO, struktur *dataset* memiliki karakteristik khusus. Berbeda dengan *dataset* untuk *image classification* yang hanya berfokus pada klasifikasi, dataset untuk *object detection* tidak hanya mencakup label kelas tetapi juga informasi spasial tentang lokasi objek dalam citra. Oleh karena itu, setiap entri dalam *dataset* ini biasanya terdiri dari citra bersama dengan *bounding box* yang menandai lokasi objek, serta label kelas yang mengidentifikasi jenis objek tersebut.

Format *dataset* mencakup citra dan anotasi yang bersesuaian. Anotasi di sini adalah, deskripsi tekstual atau *metadata* yang mengandung informasi tentang lokasi dan jenis objek dalam citra. Anotasi ini mencakup koordinat *bounding box*, label kelas, dan mungkin juga informasi tambahan seperti tingkat kepercayaan atau atribut lain dari objek.

Terdapat beberapa format *dataset* populer yang digunakan dalam deteksi objek, seperti Pascal VOC, COCO, dan YOLO. Format Pascal VOC, misalnya, menggunakan struktur XML untuk menyimpan anotasi, dengan setiap objek dalam citra diwakili oleh elemen XML yang berisi koordinat *bounding box* dan label kelas. Format COCO, di sisi lain, menggunakan struktur JSON yang lebih fleksibel, memungkinkan penyimpanan informasi yang lebih kompleks yaitu segmentasi objek. Sedangkan format YOLO lebih sederhana menyimpan anotasi dalam *file* teks biasa (ekstensi .txt), dengan setiap baris mewakili satu objek dalam citra.

2.8.2.1 Membuat *Dataset*

Pembuatan *dataset* untuk deteksi objek biasanya dimulai dengan pengumpulan citra yang berisi objek yang ingin dideteksi. Citra ini dapat diperoleh dari berbagai sumber, seperti kamera, arsip online, atau *dataset* publik yang sudah ada. Penting untuk diperhatikan bahwa isi dari *dataset* tersebut harus terdiri dari variasi citra misalnya, pencahayaan, orientasi, ukuran, dan konteks, agar model dapat belajar mengenali objek dalam berbagai kondisi.

Setelah citra dikumpulkan, langkah selanjutnya adalah menganotasi pada citra. Proses anotasi ini maksudnya memberi tanda sebuah objek dalam citra dengan informasi yang relevan, seperti lokasi (dalam bentuk *bounding box*) dan jenis objek. Proses ini biasanya dilakukan secara manual oleh *anotator* manusia yang menggunakan perangkat lunak khusus. *Anotator* akan menggambar *bounding box* di sekitar setiap objek dalam citra dan memberi label dengan kelas yang sesuai.

Anotasi harus dilakukan dengan hati-hati dan konsistensi untuk memastikan kualitas *dataset*. Kesalahan dalam anotasi dapat menyebabkan model belajar pola

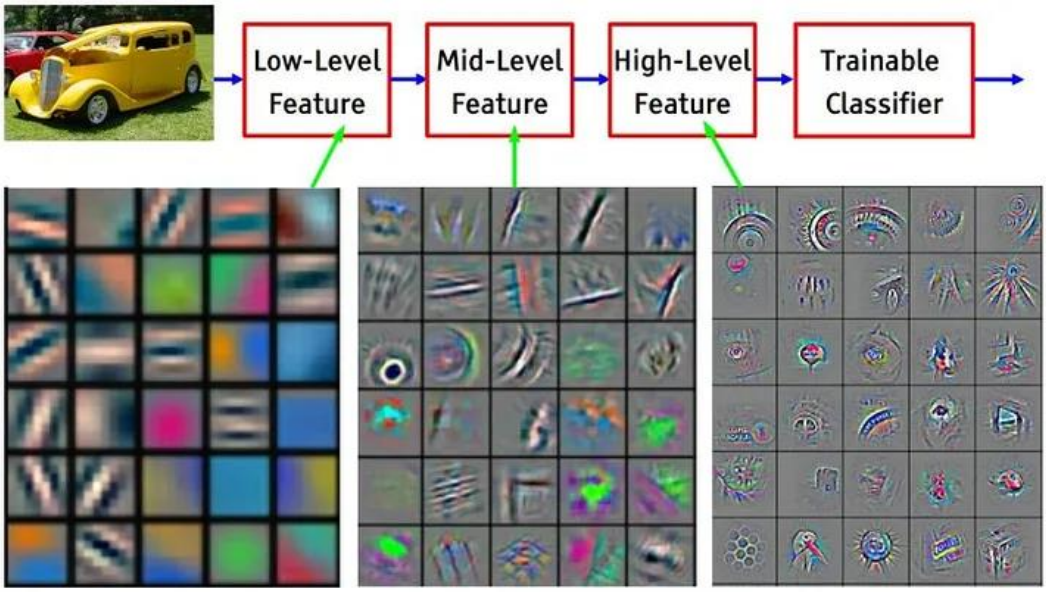
yang salah dan mengurangi kinerjanya. Oleh karena itu, seringkali diperlukan proses *review* dan validasi untuk memeriksa dan mengoreksi kesalahan dalam anotasi. Setelah anotasi selesai, citra dan anotasi yang bersesuaian biasanya disimpan dalam format yang sesuai dengan algoritma deteksi objek yang akan digunakan, seperti Pascal VOC, COCO, atau YOLO.

2.8.3 *Fine-Tuning*

Istilah *fine-tuning* dalam *deep learning* merujuk pada proses penyesuaian spesifik dari model yang telah dilatih sebelumnya untuk tugas atau aplikasi khusus. Proses ini dilakukan dengan cara melatih ulang beberapa lapisan terakhir dari model, sementara lapisan-lapisan lainnya tetap dipertahankan. Hal ini memungkinkan model untuk mempertahankan pengetahuan umum yang telah diperoleh dari data pelatihan awal, sambil menyesuaikan diri dengan karakteristik khusus dari data baru.

Dalam penelitian ini, *fine-tuning* memiliki peran yang sangat penting dalam mengoptimalkan model YOLO untuk tugas deteksi blok program dalam citra digital. Dengan menggunakan model yang telah dilatih sebelumnya, proses pengembangan menjadi lebih efisien dan efektif. *Fine-tuning* memungkinkan model untuk memanfaatkan fitur-fitur yang telah dipelajari dari data pelatihan awal, yang mencakup pola-pola umum dalam pengenalan objek, dan menyesuaikannya dengan fitur-fitur spesifik dari blok program yang akan dikenali. Pentingnya *fine-tuning* dalam penelitian ini juga terletak pada kemampuannya untuk mengurangi waktu dan sumber daya yang diperlukan untuk pelatihan, tanpa mengorbankan kualitas atau akurasi dari model yang sudah ada.

Untuk memahami lebih lanjut tentang proses *fine-tuning*, penting untuk mengenali struktur fitur dari model *deep learning* deteksi objek. Terdapat 3 bagian utama dari fitur model, yang terdiri dari *low*, *mid,* dan *high-level features*, seperti pada Gambar 2.8.



Gambar 2. 8 Ilustrasi perbedaan setiap level fitur

Gambar 2.8 mengilustrasikan perbedaan antara *low*, *mid*, dan *high-level features* dalam model *deep learning*. Setiap level memiliki karakteristik unik yang berkontribusi pada pengenalan dan interpretasi objek dalam citra. Berikut adalah penjelasan lebih rinci dari masing-masing level:

1. ***Low level features***: Pada level ini, fitur-fitur yang dihasilkan akan berkaitan dengan deteksi tepi, tekstur, dan orientasi garis dalam citra. Ini adalah representasi paling sederhana dari citra dan menjadi dasar untuk pengenalan struktur yang lebih kompleks. *Low level features* biasanya ditemukan di lapisan awal dari jaringan saraf dalam model *deep learning*, dan mereka mengenali pola-pola dasar seperti garis vertikal, diagonal, dan horizontal.
2. ***Mid level features***: Pada level ini, fitur-fitur menjadi lebih kompleks dan mencakup pengenalan bentuk, bagian dari objek, dan struktur yang lebih spesifik. Misalnya, *mid level features* dapat menggambarkan kontur atau siluet dari objek seperti wajah, mobil, atau hewan. Mulai dari sini terjadi transisi dari pola-pola dasar ke representasi yang lebih abstrak dan semantik dari objek dalam citra.

3. ***High level features***: Pada level ini, fitur-fitur sudah menggambarkan objek secara keseluruhan dan konteksnya dalam citra. Fitur-fitur ini berfokus pada pengenalan dan interpretasi objek dalam konteks yang lebih luas, menggabungkan berbagai elemen dari citra untuk menghasilkan pemahaman yang lebih menyeluruh tentang apa yang digambarkan. Ini mencakup representasi yang lebih abstrak dan kompleks dari objek, seperti identifikasi kelas objek, hubungan antar objek. *High level features* biasanya ditemukan di lapisan terakhir dari *model deep learning*.

Dalam proses *fine-tuning*, peran dari *low*, *mid*, dan *high-level features* menjadi sangat krusial. Seperti yang telah dijelaskan, *low* dan *mid-level features* merepresentasikan pola-pola dasar dan struktur yang lebih kompleks dalam citra, yang umumnya bersifat universal dan tidak tergantung pada tugas khusus. Oleh karena itu, dalam proses *fine-tuning*, lapisan yang merepresentasikan *low* dan *mid-level features* ini biasanya dibekukan atau dipertahankan, sehingga pengetahuan yang telah diperoleh dari data pelatihan awal tetap terjaga.

Sementara itu, *high-level features* merepresentasikan pemahaman yang lebih mendalam dan spesifik tentang objek dan konteks dalam citra. Oleh karena itu, karakteristik ini lebih terkait dengan tugas atau aplikasi khusus, lapisan yang merepresentasikan *high-level features* ini menjadi target utama dalam proses *fine-tuning*. Melalui pelatihan ulang hanya pada lapisan *high-level* ini, model dapat disesuaikan dengan cepat untuk mengenali fitur-fitur spesifik dari data baru tanpa mengorbankan pengetahuan umum yang telah diperoleh.

2.8.4 Evaluasi Model

Evaluasi model merupakan tahapan yang krusial dalam setiap penelitian yang melibatkan *machine learning* (ML) dan *deep learning* (DL). Tahapan ini bertujuan untuk mengukur dan menilai kinerja model yang telah dilatih, khususnya dalam mengenali dan menginterpretasi susunan blok program dalam citra digital pada penelitian ini. Melalui evaluasi yang tepat, dapat diperoleh gambaran objektif mengenai efektivitas dan efisiensi model dalam menyelesaikan tugas yang

diinginkan. Pada bagian ini akan menjelaskan macam-macam metriks evaluasi yang digunakan pada penelitian ini.

2.8.4.1 *Recall* dan *Precision*

Recall adalah salah satu metrik evaluasi yang sering digunakan dalam model ML dan DL. Metrik ini berfokus pada sejauh mana model dapat mengidentifikasi semua kasus positif yang sebenarnya dalam data. Metrik ini juga dikenal sebagai *True Positive Rate*, yaitu mengukur proporsi kasus positif yang sebenarnya yang diidentifikasi dengan benar oleh model. Perasamaan dari *recall* dapat didefinisikan sebagai:

$$Recall = \frac{TP}{TP + FN} \tag{2.7}$$

Di mana:

- TP (*True Positive)* adalah jumlah kasus di mana model dengan benar mengidentifikasi kelas positif.
- FN (*False Negative)* adalah jumlah kasus di mana model salah mengidentifikasi kelas positif sebagai kelas negatif.

Sedangkan *precision* adalah metrik untuk mengukur proporsi deteksi positif yang benar dari total deteksi positif yang dilakukan oleh model. Dengan kata lain, *precision* menilai sejauh mana model mampu mengidentifikasi objek dengan tepat, tanpa salah mengklasifikasikan objek lain. Rumus dari *precision* dapat didefinisikan sebagai:

$$Precision = \frac{TP}{TP + FP} \tag{2.8}$$

Di mana:

- FP (*False Positive)* adalah jumlah kasus di mana model salah mengidentifikasi kelas negatif sebagai kelas positif.

Precision sering digunakan bersama dengan *recall* untuk memberikan gambaran yang lebih lengkap tentang kinerja model. Sementara *Precision* berfokus pada keakuratan prediksi positif, *Recall* menilai sejauh mana model dapat mengidentifikasi semua kasus positif yang sebenarnya. Kombinasi kedua metrik ini sering digunakan dalam kurva *precision-recall*, yang menggambarkan *trade-off* antara *precision* dan *recall* untuk berbagai ambang batas. Metrik *recall* dan *precision* memiliki rentang nilai dari 0 hingga 1. Nilai 0 berarti model tidak berhasil mendeteksi satupun objek dengan benar, sedangkan nilai 1 berarti model berhasil mendeteksi semua objek dengan benar.

Sebagai contoh, terdapat model *object detection* yang mendeteksi objek apel dari sebuah citra. Model mendeteksi terdapat 8 dari 10 apel yang ada pada citra. Maka, 8 adalah *true positives* (TP), dan 2 adalah *false negatives* (FN), karena 2 apel tidak terdeteksi. *Recall* dapat dihitung sebagai:

$$Recall = \frac{8}{8+2} = \frac{8}{10} = 0.8 \tag{2.9}$$

Dari 8 apel yang dideteksi oleh model, anggaplah 2 di antaranya ternyata adalah deteksi yang salah, misalnya model salah mengidentifikasi jeruk sebagai apel. Maka, 6 adalah *true positives* (TP), dan 2 adalah *false positives* (FP). *Precision* dapat dihitung sebagai:

$$Precision = \frac{6}{6+2} = \frac{6}{8} = 0.75 \tag{2.10}$$

Dengan demikian, *recall* berfokus pada seberapa lengkap model mendeteksi apel yang ada, sedangkan *precision* fokus pada seberapa akurat deteksi apel yang dilakukan oleh model.

2.8.4.2 *Mean Average Precision*

Mean Average Precision (mAP) adalah metrik evaluasi yang lebih kompleks dibandingkan dengan Precision. Metrik evaluasi ini sering digunakan

dalam model *object detection*, karena dapat memberikan gambaran menyeluruh tentang kinerja model dalam semua kelas dan berbagai ambang batas (*thresholds*). Dengan menggabungkan dua metrik *Precision* dan *Recall*, untuk mengukur sejauh mana model mampu mendeteksi objek dengan benar tanpa banyak kesalahan positif. Rumus mAP dapat didefinisikan sebagai:

$$mAP = \frac{\sum_{i=1}^{n} AP_i}{n} \tag{2.11}$$

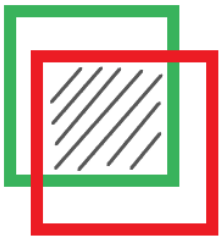
Di mana:

- n adalah jumlah kelas
- AP_i adalah *average precision* untuk kelas

Metrik ini sering digunakan dalam model *object detection*, terutama dalam situasi di mana objek yang sama dapat muncul dalam berbagai ukuran, posisi, atau orientasi dalam citra. Dengan mengukur *Precision* pada berbagai tingkat *Recall*, mAP memberikan gambaran yang lebih kaya tentang sejauh mana model dapat mendeteksi objek dalam berbagai macam kelas.

2.8.4.3 *Intersect Over Union*

Intersect over union (IoU) adalah metrik evaluasi yang sering digunakan dalam model *object detection*. Metrik ini mengukur sejauh mana dua buah *bounding boxes* tumpang tindih satu sama lain. *Bounding box* pertama biasanya dihasilkan oleh model, sedangkan *bounding box* kedua adalah nilai yang sebenarnya (*ground truth*) yang menandai lokasi objek dalam citra. IoU dapat diilustrasikan pada Gambar 2.9.



Gambar 2. 9 Ilustrasi IoU

Gambar 2.9 menggambarkan dua buah *bounding box*. *Bounding box* yang berwarna merah dapat dianggap sebagai kotak pembatas yang diprediksi oleh model, sedangkan *bounding box* hijau adalah *ground truth*. Area yang diarsir menunjukkan area persimpangan (*intersection*) antara kedua *bounding box* tersebut, sedangkan area gabungan (*union*) dari *bounding box* adalah seluruh area yang dicakup oleh kedua kotak pembatas. Rumus dari IoU dapat didefinisikan sebagai:

$$IoU = \frac{Area\ of\ Intersection}{Area\ of\ Union} \tag{2.12}$$

Nilai IoU berkisar antara 0 hingga 1. Nilai 0 berarti tidak ada tumpang tindih antara dua *bounding box*, sedangkan nilai 1 berarti kedua *bounding box* tumpang tindih sempurna. IoU sering digunakan dalam evaluasi model deteksi objek untuk mengukur sejauh mana prediksi model sesuai dengan *ground truth*.