BAB II
TINJAUAN PUSTAKA

2.1 Penelitan-Penelitian Sebelumnya

Penelitian terdahulu adalah upaya peneliti untuk mencari perbandingan dan selanjutnya untuk menemukan inspirasi baru untuk peneltiain selanjutnya di samping itu kajian terdahulu membantu penelitian dapat memposisikan penelitian serta menujukkan orsinalitas dari penelitian. Pada bagaian ini peneliti mencamtumkan berbagai hasil penelitian terdahulu terkait dengan penelitian yang hendak dilakukan, kemudian membuat ringkasannya, baik penelitian yang sudah terpublikasikan atau belum terpublikasikan. Berikut merupakan penelitian terdahulu yang masih terkait dengan tema yang penulis kaji.

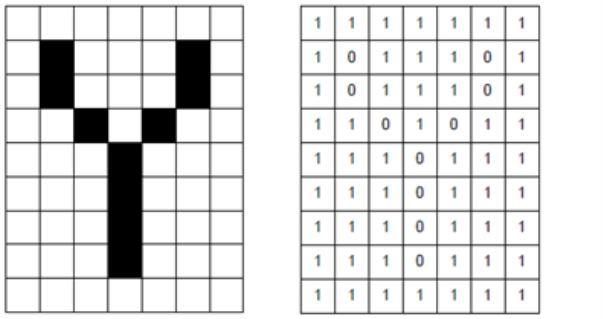
Pertama, penelitian yang dilakukan oleh Yuda Puspito, F. X. Arinto Setyawan, dam Helmy Fitriawan (2018) dalam penelitiannya yang berjudul "Deteksi Posisi Plat Nomor Kendaraan Menggunakan Metode *Transformasi Hough Dan Hit Or Miss*". Penelitian ini dikembangkan sebuah sistem pendeteksi posisi plat nomor kendaraan yang ditampilkan pada GUI Matlab. Pendeteksian posisi plat nomor kendaraan menggunakan dua metode, yaitu metode *transformasi hough dan transformasi hit or miss*. Tahap pengolahan citra yang digunakan meliputi: binerisasi, aras keabuan, deteksi tepi, pemotongan citra, *filtering*, dan *resizing*. Keefektifan sistem ini diukur dengan perhitungan terhadap nilai perolehan (*recall*) dan nilai ketepatan (*precision*). Berdasarkan hasil penelitian didapatkan bahwa sistem berhasil mendeteksi posisi plat nomor kendaraan dengan tingkat keberhasilan pendeteksian sebesar 76% untuk nilai *threshold* 0,75, 72% untuk nilai *threshold* 0,8 dan 48% untuk nilai *threshold* 0,85. Hasil penelitian juga menunjukkan nilai rata-rata recall sebesar 54% untuk nilai threshold 0,75, 50% untuk nilai *threshold* 0,8 dan 40% untuk nilai *threshold* 0,85, sedangkan nilai rata-rata *precision* sebesar 14% untuk nilai *threshold* 0,75, 14% untuk nilai *threshold* 0,8 dan 12% untuk nilai *threshold* 0,85 [7].

Penelitian selanjutnya, yaitu dengan judul "Pendeteksian Plat Nomor Kendaraan Menggunakan Algoritma *You Only Look Once* V3 dan *Tesseract*" oleh Muhamad Rizky Fauzan dan Ari Purno Wahyu W. Penelitian tersebut membahas sistem yang dapat mendeteksi plat nomor kendaraan dan mengenali karakter dari plat nomor kendaraan tersebut. Sistem tersebut menggunakan algoritma *You Only Look Once* V3 sebagai algoritma pendeteksi objek dan *Tesseract Optical Character Recognition* sebagai pendeteksi teks dalam gambar. Hasil dari penelitian tersebut yaitu sistem yang digunakan untuk mendeteksi objek menggunakan YOLO v3 dapat ditampilkan pada layar laptop secara langsung. Kemudian untuk mendeteksi dan mengenali karakter teks menggunakan *tesseract* OCR dilakukan dengan merekam objek melalui *webcam* dan menampilkan hasilnya secara *real-time* [8].

2.2 Image Processing

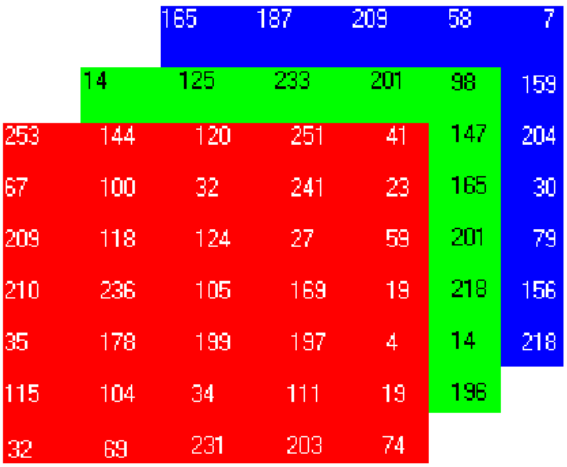
Image processing adalah penggunaan komputer digital untuk memproses gambar digital melalui algoritma [9]. Sebagai sub-kategori atau bidang pemrosesan sinyal digital, pemrosesan gambar digital memiliki banyak keunggulan dibandingkan pemrosesan gambar analog. Hal ini memungkinkan algoritma yang jauh lebih luas untuk diterapkan pada data input dan dapat menghindari masalah seperti penumpukan *noise* dan distorsi selama pemrosesan. Karena gambar didefinisikan lebih dari dua dimensi (mungkin lebih) pemrosesan gambar digital dapat dimodelkan dalam bentuk sistem multidimensi [10].

Citra digital merupakan suatu *array* dari bilangan *real* atau *complex number* yang diwakili oleh sejumlah bit (*pixel*) yang terbatas [11]. Maka dari itu, dalam representasi citra kita lebih memperhatikan karakterisasi kuantitas yang diwakili oleh setiap elemen gambar yang biasa disebut dengan *pixel*. Citra digital dapat dimanipulasi menggunakan matriks, karena gambar dapat dipecah menjadi beberapa bagian bit (*pixel*) seperti gambar 2.1 berikut.



Gambar 2.1 Representasi Citra Dalam Matriks

Pada gambar 2.1 di atas, citra digital dapat direpresentasikan dengan menggunakan matriks dua dimensi. Namun, pada citra digital berwarna informasi warna ini dipecah dalam tiga komponen warna yang biasa disebut RGB (*Red Green Blue*), dari ketiga komponen utama tersebut apabila dikombinasikan dapat menciptakan warna apapun. Oleh karena itu, citra digital berwarna memiliki bentuk matriks tiga dimensi yang mana setiap dimensinya mewakili warna RGB seperti yang diilustrasikan pada gambar 2.2 di bawah.



Gambar 2.2 Matriks RGB 3-dimensi

Pada gambar 2.2, nilai warna pada *pixel* dari masing-masing komponen berkisar 0 sampai dengan 255 dengan tipe data integer (angka tanpa desimal).

Apabila pada *pixel* dari setiap komponen warna diberi angka 255 maka akan menghasilkan warna putih, apabila diberikan angka 0 semua akan mendapatkan warna hitam.

2.2.1 Filtering

Filtering adalah proses mengubah nilai piksel dalam citra dengan menerapkan operasi tertentu menggunakan *filter* atau *kernel*. Tujuan utamanya adalah untuk menghilangkan *noise*, menghaluskan citra, atau menonjolkan fitur-fitur tertentu. Dalam *filtering*, setiap piksel dalam citra diubah dengan mempertimbangkan nilai piksel di sekitarnya. *Filter* dapat berupa matriks kecil yang ditempatkan di atas piksel dan operasi matematis dilakukan antara piksel citra dan elemen *filter*.

Kernel dalam konteks pengolahan citra digital merujuk pada matriks atau himpunan nilai yang digunakan untuk menerapkan operasi konvolusi atau *filtering* pada citra. *Kernel* ini berfungsi sebagai pola yang akan digeser di atas seluruh citra, dan pada setiap posisi, operasi matematis akan dijalankan antara nilai piksel dalam *kernel* dan piksel-piksel di bawahnya. *Kernel* menentukan bagaimana operasi konvolusi akan mempengaruhi citra. Berikut beberapa contoh *kernel* yang umum digunakan dalam pengolahan citra digital:

- ***Gaussian Kernel***: *Gaussian Kernel* digunakan untuk operasi penghalusan atau pengurangan *noise* dalam citra. Ini membantu mengurangi efek *noise* dan menghaluskan perubahan tajam dalam intensitas citra. *Kernel* ini biasanya memiliki distribusi *Gaussian* yang menempatkan bobot lebih besar pada piksel di tengah *kernel* dan bobot lebih rendah pada piksel di luar *kernel*. Ukuran dan *sigma* (standar deviasi) *kernel* dapat diatur untuk mengendalikan seberapa banyak penghalusan yang diterapkan.
- ***Sobel Kernel***: *Sobel Kernel* digunakan untuk deteksi tepi (*edge detection*) dalam citra. Terdapat dua *kernel Sobel,* yaitu untuk deteksi tepi secara horizontal dan vertikal. *Sobel Kernel* untuk deteksi tepi horizontal akan mengidentifikasi perubahan intensitas vertikal dalam citra, sedangkan *kernel*

Sobel untuk deteksi tepi vertikal akan mengidentifikasi perubahan intensitas horizontal. Kedua *kernel* ini membantu dalam menemukan di mana tepi-tepi citra terletak.

- ***Prewitt Kernel***: *Prewitt kernel* juga digunakan untuk deteksi tepi dalam citra. Dalam deteksi tepi, *kernel Prewitt* memiliki pola serupa dengan *Sobel*, tetapi bobot dalam *kernel Prewitt* lebih sederhana, yaitu hanya -1, 0, dan 1. *Kernel* ini juga memiliki varian untuk deteksi tepi dalam arah horizontal dan vertikal.
- ***Laplacian Kernel***: *Laplacian Kernel* digunakan untuk meningkatkan fitur tepi dalam citra. Ini bertujuan untuk meningkatkan perbedaan intensitas di sekitar tepi. *Laplacian kernel* memiliki pusat positif dan tetangga negatif, yang menciptakan efek penguatan kontras pada tepi dan memudarkan wilayah homogen.
- ***Median Kernel***: *Median Kernel* digunakan untuk menghilangkan *noise* tipe "*salt-and-pepper*" dari citra. Dalam operasi ini, nilai piksel di tengah *kernel* diganti dengan nilai median dari piksel-piksel di dalam *kernel*. Ini membantu menghilangkan piksel-piksel yang ekstrem (terlalu terang atau terlalu gelap) yang mungkin muncul sebagai *noise*.
- ***Sharpening Kernel***: *Sharpening Kernel* digunakan untuk meningkatkan detail dalam citra. *Kernel* ini mengambil perbedaan antara piksel di tengah dan piksel di sekitarnya, yang akan menguatkan perbedaan tajam dalam intensitas. Hal ini menghasilkan efek peningkatan kontras di sepanjang tepi dan detail dalam citra.
- ***Embossing Kernel***: *Embossing Kernel* menciptakan efek tiga dimensi pada citra dengan menyoroti perbedaan intensitas antara piksel di tengah *kernel* dan piksel-piksel di sekitarnya. Ini menghasilkan ilusi tiga dimensi dan memberikan tampilan berbeda pada citra.

2.2.2 Convolution

Convolution (konvolusi) pada pengolahan citra adalah operasi matematis yang digunakan untuk menerapkan *filter* atau *kernel* pada citra dengan tujuan

menghasilkan citra baru yang telah difilter. Konsep konvolusi melibatkan penggeseran *filter* di atas citra dan pada setiap posisi, melakukan perkalian dan penjumlahan antara nilai piksel dalam citra dan nilai yang sesuai dalam *filter*. Hasil dari operasi konvolusi ini adalah citra yang telah diubah sesuai dengan pola atau karakteristik yang ditentukan oleh *filter*. Penggunaan konvolusi dalam pengolahan citra memiliki beberapa tujuan, termasuk:

- **Penghalusan (*Smoothing*)**: Konvolusi dengan *kernel* rata-rata atau *Gaussian* dapat menghaluskan citra dengan mengurangi perbedaan tajam antara piksel-piksel.
- **Deteksi Tepi (*Edge Detection*)**: *Kernel Sobel* atau *Prewitt* digunakan untuk mengidentifikasi perubahan tajam dalam intensitas piksel, yang menunjukkan adanya tepi dalam citra.
- **Peningkatan (*Enhancement*)**: Konvolusi dengan *kernel* tertentu dapat meningkatkan kontras dan mempertajam fitur-fitur dalam citra.
- **Pengurangan *Noise***: Beberapa *kernel*, seperti *median filter*, dapat digunakan untuk menghilangkan *noise* dari citra dengan mengganti nilai piksel dengan nilai median di sekitarnya.
- **Transformasi dan Analisis**: Konvolusi juga digunakan dalam transformasi *Fourier* dan operasi matematis lainnya untuk analisis frekuensi citra.

2.2.3 Segmentation

Segmentasi (*segmentation*) pada pengolahan citra adalah proses membagi citra menjadi wilayah-wilayah yang memiliki karakteristik serupa atau objek-objek yang dapat diidentifikasi. Tujuannya adalah untuk memisahkan dan mengidentifikasi bagian penting dalam citra. Teknik segmentasi digunakan untuk mengambil bagian dari citra yang menarik minat kita, memisahkan objek dari latar belakang, atau mengidentifikasi bagian yang memiliki makna khusus [12]. Berikut adalah metode-metode dari segmentasi:

- ***Thresholding***: Ini adalah metode paling sederhana di mana piksel-piksel dalam citra diubah menjadi hitam atau putih berdasarkan nilai ambang

tertentu. Semua piksel dengan intensitas di atas ambang menjadi putih dan yang di bawah ambang menjadi hitam. Ini sering digunakan untuk segmentasi objek yang memiliki perbedaan tajam dalam intensitas dengan latar belakang.

- ***Clustering***: Teknik ini menggunakan metode *clustering* untuk mengelompokkan piksel-piksel yang serupa dalam kelompok atau kluster. Salah satu metode yang umum adalah *K-means clustering*. *Clustering* berguna ketika objek-objek dalam citra tidak memiliki perbedaan intensitas yang jelas.
- **Deteksi Tepi dan Garis**: Segmentasi dapat dilakukan dengan mendeteksi tepi dan garis dalam citra. Ini mencakup penggunaan *filter* tepi seperti *Sobel* atau *Canny* untuk menyoroti perubahan tajam dalam intensitas piksel, yang sering mengindikasikan tepi objek.
- ***Region Growing***: Metode ini dimulai dengan satu piksel atau wilayah yang telah ditentukan dan secara iteratif menambahkan piksel-piksel yang memiliki karakteristik yang serupa. Ini memungkinkan segmentasi berdasarkan wilayah yang memiliki konsistensi dalam fitur tertentu.
- ***Active Contour Model (Snake)***: Ini adalah pendekatan yang menggunakan kurva berbentuk seperti ular (*snake*) yang dapat mengubah bentuk dan bergerak untuk menyesuaikan batas objek yang diinginkan dalam citra.

2.3 Artificial Intelligence

Artificial Intelligence (AI) adalah cabang ilmu komputer yang bertujuan untuk menciptakan mesin atau sistem yang memiliki kemampuan untuk melakukan tugas yang membutuhkan kecerdasan manusia. Ini mencakup berbagai teknik dan metode untuk mengembangkan komputer yang dapat belajar, merencanakan, memecahkan masalah, beradaptasi, dan bahkan berinteraksi dengan manusia. AI mencakup berbagai sub-bidang seperti *machine learning*, *natural language processing*, *computer vision*, *robotics*, dan banyak lagi.

Salah satu karakteristik penting dari AI adalah sifat multidisiplinnya. AI memiliki hubungan erat dengan ilmu komputer, tetapi pengembangan proyek-

proyek AI juga memerlukan pemahaman dari logika, linguistik, ilmu kognitif, dan sebagainya. Selain itu, AI digunakan dalam berbagai bidang aplikasi, seperti kedokteran, industri, perbankan, keuangan, dan lain-lain[13].

2.4 Machine Learning

Machine Learning (Pembelajaran Mesin) adalah sub-bidang dalam kecerdasan buatan (AI) yang berfokus pada pengembangan teknik dan algoritma yang memungkinkan komputer untuk belajar dari data dan meningkatkan kinerjanya dalam suatu tugas tertentu. Konsep inti dari *Machine Learning* adalah kemampuan komputer untuk "belajar" pola dari data, sehingga mereka dapat membuat prediksi atau keputusan berdasarkan data yang belum pernah dilihat sebelumnya [14]. Dalam *Machine Learning*, ada beberapa pendekatan utama, yaitu:

- ***Supervised Learning***: Model dibuat untuk memahami hubungan antara input dan output dengan memanfaatkan data latihan yang sudah dilabeli. Tujuannya adalah untuk memetakan input ke output yang benar. Contoh aplikasi termasuk klasifikasi (mengategorikan data ke dalam kelas tertentu) dan regresi (memprediksi nilai berdasarkan input).
- ***Unsupervised Learning***: Dalam jenis pembelajaran ini, model belajar menemukan pola-pola dalam data yang tidak memiliki label. Ini sering digunakan untuk pengelompokan data (*clustering*) atau reduksi dimensi.
- ***Reinforcement Learning***: Model belajar melalui interaksi dengan lingkungannya dan mengambil tindakan yang akan mengoptimalkan hadiah atau penghargaan jangka panjang. Ini cocok untuk pengambilan keputusan berurutan, seperti dalam permainan atau robotika.
- ***Semi-Supervised Learning***: Model belajar dari dataset yang sebagian besar tidak dilabeli (*unlabelled data*) dan sebagian kecil di antaranya dilabeli (*labelled data*). Pendekatan ini dimaksudkan untuk mengatasi keterbatasan dalam memperoleh data yang dilabeli, yang seringkali mahal dan memakan waktu.

- ***Self-Supervised Learning***: *Self-Supervised Learning* adalah bagian dari metode *Unsupervised Learning*. *Self-Supervised Learning* merujuk pada metode pembelajaran di mana *ConvNets* dilatih secara eksplisit dengan label yang dihasilkan secara otomatis. Ulasan ini hanya berfokus pada metode *Self-Supervised Learning* untuk pembelajaran fitur visual dengan *ConvNets* di mana fitur-fitur tersebut dapat ditransfer ke berbagai tugas komputer visi yang berbeda [14].

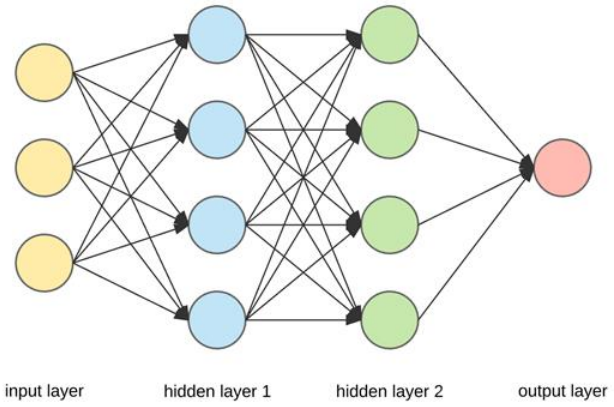
2.5 Deep Learning

Deep Learning (DL) merupakan sub-bidang *machine learning* yang berhubungan dengan algoritma yang terinspirasi oleh struktur dan fungsi otak manusia yang disebut jaringan saraf tiruan (JST) atau dalam Bahasa Inggrisnya yaitu *Artificial Neural Network* (ANN). Dengan kata lain, hal ini mencerminkan fungsi otak manusia. Algoritma *deep learning* mirip dengan bagaimana sistem saraf terstruktur di mana setiap neuron saling terhubung dan menyampaikan informasi [15].

Deep learning adalah sebuah teknologi yang mampu bekerja menggunakan beberapa algoritma tertentu. Setiap algoritma *deep learning* memiliki kegunaan dan keunggulannya yang berbeda-beda [16]. Oleh karena itu, pemilihan jenis algoritma *deep learning* sangatlah penting karena harus sesuai dengan kebutuhan.

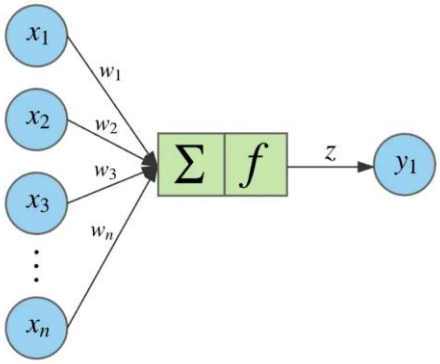
2.5.1 Artificial Neural Network (ANN)

ANN (*Artificial Neural Network*) adalah model matematika terinspirasi oleh struktur jaringan saraf dalam otak manusia. Tujuan utama ANN adalah meniru kemampuan otak dalam memproses informasi dan belajar dari data. Model ini terdiri dari banyak unit pemrosesan sederhana yang disebut "*neuron*" atau "*node*" yang saling terhubung dan dapat bekerja secara paralel untuk memproses data. Ilustrasi dari arsitektur ANN dapat dilihat pada gambar 2.3.



Gambar 2.3 Arsitektur ANN

Artificial Neural Network (ANN) adalah jaringan komputasi yang terinspirasi dari sistem saraf manusia. ANN merupakan *multi-layer fully-connected neural networks* [17], seperti yang terlihat pada gambar 2.3. ANN terdiri dari *input layer* (lapisan input), beberapa *hidden layer* (lapisan tersembunyi), dan *output layer* (lapisan output). Setiap *node* (simpul) dalam satu lapisan terhubung ke setiap *node* lain di lapisan berikutnya. Kita dapat membuat jaringan lebih dalam dengan meningkatkan jumlah *hidden layer*. Jika kita memperbesar ke salah satu *hidden* atau *output node*, maka akan seperti pada gambar 2.4 di bawah.



Gambar 2.4 Perceptron

Pada gambar 2.4, terlihat bahwa dari *node* sebelumya akan mengirimkan nilai (*x*) yang mana nilai tersebut akan dikalikan oleh *weight* (*w*) dan melalui *non-linear activation function* sehingga menghasilakn *output z* yang akan dikirim ke *node*

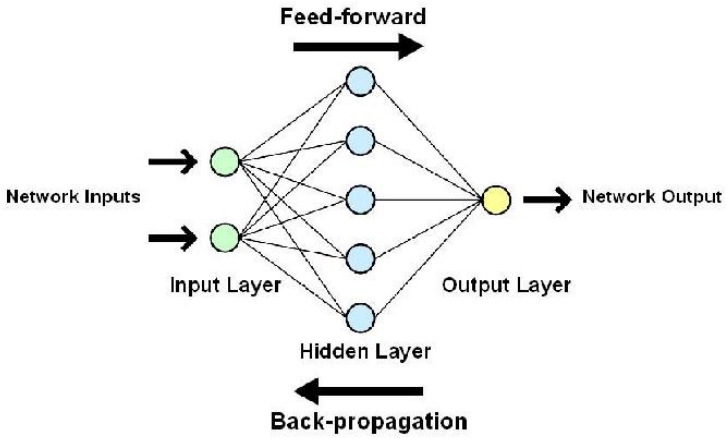
berikutnya sebagai input, yaitu *y* [17]. Sinyal mengalir dari kiri ke kanan, dan *output* akhir dihitung dengan melakukan prosedur ini untuk semua *node*. Melatih *deep neural network* berarti mempelajari *wight* (bobot) yang terkait dengan semua sisi. Kejadian tersebut dikenal sebagai *perceptron*.

$$z = f(b + x \cdot w) = f\left(b + \sum_{i=1}^{n} x_1 w_1\right) \tag{2. 1}$$

$$x \in d_{1\times n}, w \in d_{n\times 1}, b \in d_{1\times 1}, z \in d_{1\times 1}$$

Persamaan untuk *node* yang diberikan terlihat seperti pada persamaan 2.1. Jumlah *weight* dari inputnya melewati *non-linear activation function*. Hal itu dapat direpresentasikan sebagai *vector dot product*, di mana *n* adalah jumlah input untuk *node* dan *b* merupakan *bias*. *Bias* merupakan masukan untuk semua *node* dan selalu bernilai 1. Hal itu memungkinkan untuk menggeser hasil *activation function* ke kiri atau ke kanan. Hal tersebut juga membantu model untuk melatih ketika semua fitur input adalah 0 [17].

ANN juga dapat dikatakan sebagai *perceptron multi-layer* yang disebut *feed-forward neural network* karena terdiri dari *hidden layer*, *bias unit*, *neuron* (*input*, *output*, dan *perceptron*), *weight*, serta *activation function*. Setiap ANN bertujuan untuk memperkirakan fungsi *f* yang memberikan perkiraan untuk satu set *output* pada setiap satu set *input* yang diberikan. ANN yang disebutkan di atas disebut *feed-forward* (umpan-maju) karena tidak ada *back-propagation* (umpan-balik) yang diberikan kembali ke *input*. Ilustrasi dari *feed-forward* dan *back-propagation* terlihat pada gambar 2.5.



Gambar 2.5 Feed-forward dan Back-propagation

Pada gambar 2.5 terlihat bahwa terdapat perbedaan antara *feed-forward* dengan *back-propagation*, perbedaannya yaitu arah dari pergerakan sinyal *input*. Di mana pada *feed-forward* sinyal *input* bergerak menuju *hidden layer* kemudian bergerak menuju *output layer*. Sedangkan pada *back-propagation*, sinyal *input* pada awalnya bergerak sama seperti *feed-forward*, namun setelah setiap *forward pass* (bergerak maju) melalui jaringan, *back-propagation* melakukan *back pass* (bergerak mundur) sambil menyesuaikan parameter (*weight* dan *bias*) pada model.

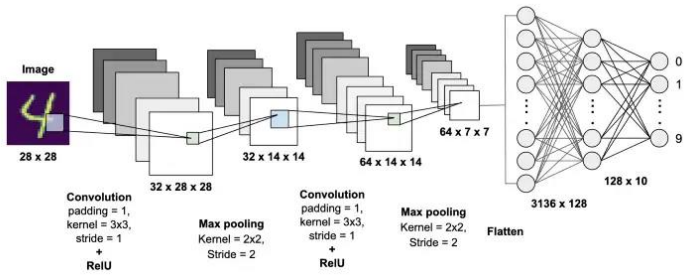
Pada ANN, penggunaan *back-propagation* yaitu sebagai algoritma untuk *gradient descent* dengan memerhatikan *weight* (bobot). *Output* yang diinginkan akan dibandingkan dengan *output* yang dihasilkan oleh sistem. Setelah itu, sistem akan melakukan *tuning* dengan cara menyesuaikan *weight* koneksi yang bertujuan untuk memperkecil perbedaan/selisih antara kedua *output* tersebut [18].

2.5.2 Convolutional Neural Network (CNN)

Convolutional Neural Network (CNN) merupakan salah satu jenis *neural network* yang biasa digunakan pada data *image* (gambar). Oleh karena itu, CNN dapat digunakan untuk mendeteksi dan mengenali objek pada sebuah *image* [19].

Secara garis besar, arsitektur CNN tidak jauh beda dengan *neural network* pada umumnya. CNN terdiri dari *neuron* yang memiliki *weight, bias* dan *activation function* [19]. Salah satu perbedaan utama yaitu *neuron-neuron* yang merupakan *layer-layer* di dalam CNN terdiri dari *neuron-neuron* yang diorganisasikan ke dalam tiga dimensi, dimensi spasial dari input (tinggi dan lebar) dan kedalaman. Kedalaman tersebut tidak mengacu pada jumlah total lapisan dalam ANN, tetapi dimensi ketiga dari *activation volume*. Tidak seperti ANN pada umumnya, *neuron-neuron* di dalam setiap *layer* yang diberikan hanya akan terhubung ke wilayah kecil (*small region*) dari *layer* sebelumnya [20].

CNN terdiri dari tiga jenis *layer*, yaitu: *convolutional layer* (*Conv layer*), *pooling layer* dan *fully-connected layer* yang terhubung sepenuhnya. Ketika lapisan-lapisan ini ditumpuk, maka sebuah arsitektur CNN telah terbentuk [20]. Contoh dari arsitektur CNN untuk klasifikasi MNIST dapat dilihat pada gambar 2.6 berikut.



Gambar 2.6 Contoh arsitektur CNN

Fungsionalitas dasar dari contoh arsitektur CNN pada gambar 2.6 dapat dipecah menjadi empat area utama, yaitu:

1. ***Input layer***. Seperti yang ditemukan dalam bentuk ANN lainnya, lapisan input akan menampung nilai *pixel* dari gambar.
2. ***Convolutional layer***. Lapisan ini akan menentukan *output neuron-neuron* yang terhubung ke *local region* dari *input* melalui perhitungan *scalar product* antara *weights* dan daerah (*region*) yang terhubung ke

volume input. *Rectified Linear Unit* (ReLU) bertujuan untuk menerapkan *activation function* '*elementwise*' seperti *sigmoid* ke *output* dari aktivasi yang dihasilkan oleh lapisan sebelumnya.

3. ***Pooling layer***. Lapisan ini hanya akan melakukan *downsampling di* sepanjang dimensi spasial dari *input* yang diberikan, selanjutnya mengurangi jumlah parameter dalam aktivasi tersebut.
4. ***Fully-connected layer***. Lapisan ini akan melakukan tugas yang sama seperti dalam ANN pada umumnya dan mencoba untuk menghasilkan *class score* dari *activation*, yang akan digunakan untuk klasifikasi. ReLU juga dapat digunakan di antara lapisan-lapisan ini, untuk meningkatkan kinerja.

Dalam CNN juga terdapat operasi konvolusi, di mana operasi konvolusi pada CNN merupakan langkah inti dalam proses pengolahan data berdimensi tinggi, terutama dalam konteks pemrosesan citra. Hal ini melibatkan penggunaan *kernel* atau *filter* untuk mengekstrak fitur-fitur penting dari data input [21]. Berikut adalah penjelasan lebih rinci tentang operasi konvolusi pada CNN:

1. **Input Data**: Data input pada CNN biasanya berupa citra atau matriks berdimensi tinggi lainnya. Data ini dapat memiliki tiga dimensi dalam kasus citra warna (lebar, tinggi, dan saluran warna) atau dua dimensi dalam kasus citra hitam-putih (lebar dan tinggi).
2. ***Kernel* (*Filter*)**: Setiap lapisan dalam CNN memiliki sejumlah *kernel*. *Kernel* ini berukuran relatif kecil dan bergerak melalui data input untuk mengekstrak fitur-fitur khusus. Setiap *kernel* adalah matriks bobot yang digunakan untuk mengalikan nilai-nilai dalam data input yang tumpang tindih dengan nilai bobotnya.
3. **Penggeseran (*Sliding*)**: *Kernel* mulai digeser dari sudut kiri atas data input dan bergerak sepanjang input dengan langkah yang ditentukan

oleh *stride*. *Stride* adalah jumlah langkah yang diambil oleh *kernel* saat bergerak.

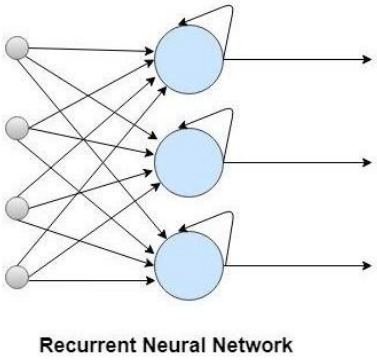
4. **Perhitungan Konvolusi**: Pada setiap posisi, perhitungan konvolusi dilakukan. Hal ini melibatkan perkalian elemen-elemen *kernel* dengan nilai-nilai terkait dalam data input yang tumpang tindih, dan hasil perkalian tersebut dijumlahkan untuk menghasilkan nilai tunggal. Nilai ini kemudian ditempatkan di posisi yang sesuai dalam output.
5. **Output**: Proses konvolusi ini menghasilkan output dalam bentuk "*feature map*" atau "*activation map*". Setiap elemen dalam *feature map* mencerminkan bagaimana *kernel* merespons atau mendeteksi fitur tertentu dalam data input. Misalnya, *kernel* awal mungkin akan mendeteksi tepi atau garis-garis sederhana, sementara *kernel* yang lebih dalam dapat mendeteksi pola-pola yang lebih kompleks.
6. ***Padding***: *Padding* (pengisian) dapat ditambahkan pada data input sebelum operasi konvolusi untuk mempertahankan dimensi output. Padding ini sering kali berupa nilai nol yang ditambahkan di sekitar tepi input.
7. **Fungsi Aktivasi**: Setelah konvolusi dilakukan, biasanya diikuti oleh fungsi aktivasi seperti ReLU (*Rectified Linear Unit*). Fungsi ini diterapkan pada setiap elemen dalam feature map untuk menghilangkan nilai negatif dan memperkenalkan non-linearitas.
8. ***Pooling***: Setelah operasi konvolusi dan aktivasi, lapisan *pooling* (seperti *max pooling* atau *average pooling*) mungkin diterapkan untuk mengurangi dimensi dan kompleksitas komputasi, serta mempertahankan fitur penting.

2.5.3 Recurrent Neural Network (RNN)

ANN dengan koneksi berulang disebut Recurrent Neural Network (RNN), yang mampu memodelkan data sekuensial untuk *sequence*

recognition dan prediksi [22]. RNN terbuat dari *hidden state* berdimensi tinggi dengan dinamika non-linear. Struktur *hidden state* bekerja sebagai memori jaringan dan keadaan *hidden layer* pada suatu waktu dikondisikan pada keadaan sebelumnya. Struktur ini memungkinkan RNN untuk menyimpan, mengingat, dan memproses sinyal kompleks sebelumnya untuk periode waktu yang lama. RNN dapat memetakan *input sequence* ke *output sequence* pada langkah waktu saat ini dan memprediksi urutan pada langkah waktu berikutnya [22].

Fitur mendasar dari *recurrent neural network* (RNN) adalah bahwa jaringan mengandung setidaknya satu *feed-back connection*, sehingga *activation* dapat mengalir dalam satu lingkaran. Hal itu memungkinkan jaringan untuk melakukan pemrosesan temporal dan mempelajari urutan (*sequence*), misalnya, melakukan pengenalan/produksi ulang urutan atau asosiasi/prediksi temporal [23]. Contoh arsitektur RNN dapat dilihat pada gambar 2.7.



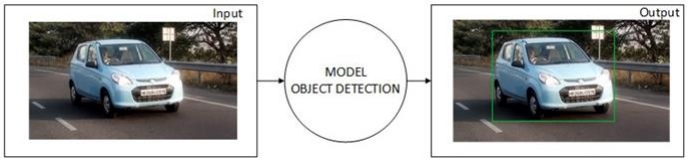
Gambar 2.7 Contoh arsitektur RNN

Dari gambar 2.7, terlihat bahwa RNN memiliki relasi berulang pada *hidden state*. Kendala perulangan ini memastikan penangkapan informasi sekuensial dalam *input data*. Dapat dikatakan bahwa RNN memiliki memory yang berisikan hasil rekaman informasi yang dihasilkan sebelumnya.

2.6 Object Detection

Object Detection atau deteksi objek adalah proses menemukan *instance* objek dari kelas tertentu, seperti wajah, mobil, dan pohon, dalam gambar atau video. Tidak seperti klasifikasi, deteksi objek dapat mendeteksi banyak objek, serta lokasinya di gambar. Detektor objek akan mengembalikan daftar objek yang terdeteksi dengan informasi kelas objek, probabilitas, dan koordinatnya untuk setiap objek [24].

Tugas dari *Object Detection* yaitu bagaimana membuat mesin dapat mengenali beberapa objek dan menentukan posisi objek-objek tersebut di dalam sebuah gambar. Konsep *Object Detection* secara sederhana yaitu dengan melakukan *scanning* pada seluruh bagian gambar dan menentukan mana yang objek dan mana yang bukan objek (*background*) [25]. Ilustrasi *object detection* dapat dilihat pada gambar 2.8 berikut.



Gambar 2.8 Ilustrasi Object Detection

Gambar 2.8 di atas merupakan ilustrasi dari *Object Detection*, di mana sistem diberi *task* untuk mendeteksi sebuah objek berupa mobil. Dari ilustrasi di atas, sistem akan menerima inputan berupa gambar yang di dalamnya terdapat beberapa objek seperti pohon, rambu lalu lintas, mobil dan lainnya. Selanjutnya diteruskan ke dalam sebuah model dari sistem yang telah di-*training* untuk mendeteksi sebuah mobil. Terakhir, outputnya berupa koordinat yang merepresentasikan posisi objek dalam gambar. Koordinat ini akan membentuk sebuah kotak pembatas (*bounding box*) di sekitar objek [25].

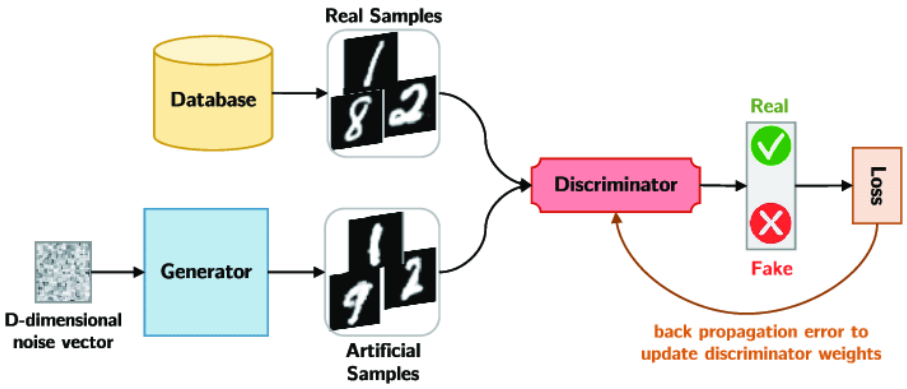
Beberapa model yang digunakan dalam deteksi objek pada gambar maupun video diantaranya yaitu: *Faster* R-CNN, YOLO, *MobileNet*, *Mask* R-CNN, *Fast* R-FCN, dan lainnya. Kebanyakan model *object detection* memanfaatkan wilayah pada

gambar dalam menentukan objek. Sehingga model tidak melihat gambar secara keseluruhan [25].

2.7 Generative Adversarial Networks

Generative Adversarial Networks (GANs) adalah kelas algoritma AI yang digunakan dalam *self-supervised machine learning*, yang diimplementasikan oleh sistem dua *neural network*s yang saling berkompetisi dalam kerangka *zero-sum game*. *Generative modelling* adalah tugas *self-supervised leaning* dalam *machine learning* yang melibatkan secara otomatis menemukan dan mempelajari keteraturan atau pola dalam *input data* sedemikian rupa sehingga model dapat digunakan untuk men-*generate* atau menghasilkan contoh baru yang secara masuk akal dapat diambil dari *dataset* asli [26].

GANs adalah cara cerdas untuk melatih *generative model* dengan membingkai masalah sebagai masalah *supervised learning* dengan dua sub-model: model generator (*generator model*) yang kita latih untuk menghasilkan contoh baru, dan model pembeda (*discriminator model*) yang mencoba mengklasifikasikan contoh sebagai gambar asli (dari domain) atau palsu (yang dihasilkan). Kedua model dilatih bersama dalam *zero-sum game*, *adversarial*, sampai model diskriminator tertipu sekitar separuh waktu, yang berarti model generator menghasilkan contoh yang masuk akal. Arsitektur GANs dapat dilihat pada gambar 2.9 di bawah.

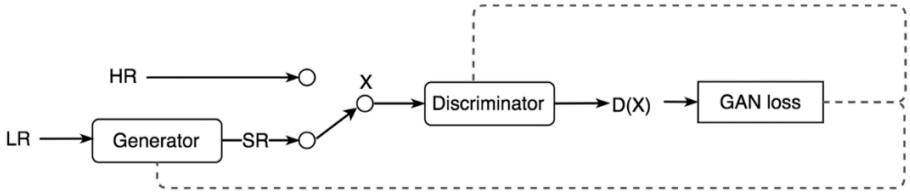


Gambar 2.9 Arsitektur GANs

Pada gambar 2.9, dua model *neural networks* tersebut dikenal sebagai *Generator model* dan *Discriminator model*. Yang mana input dari *generator* yaitu berupa *noise vector*, yang nantinya akan menghasilkan sample buatan (gambar palsu). Sedangkan, *discriminator* menerima sampel dari dataset (gambar asli) dan output dari *generator* (gambar palsu), kemudian *discriminator* memiliki tugas untuk membedakan antara kedua gambar tersebut [26].

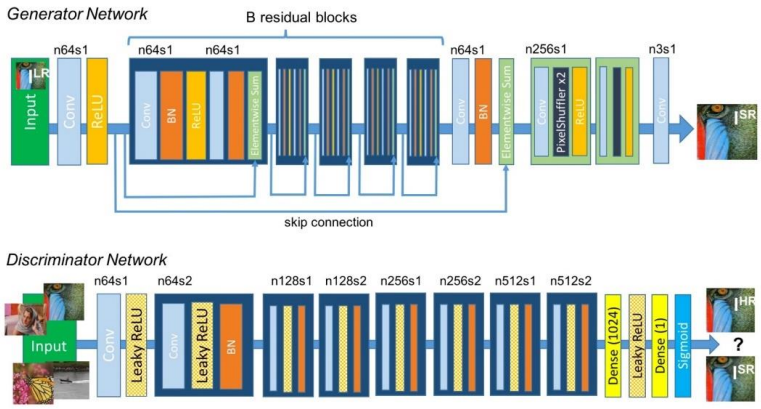
2.7.1 Super-Resolution GAN (SRGAN)

Super-resolution GAN (SRGAN) menerapkan *deep network* yang dikombinasikan dengan *adversary network* untuk menghasilkan gambar dengan resolusi yang lebih tinggi. Selama proses *training*, gambar resolusi tinggi atau *high resolution* (HR) di-*downsampling* ke gambar resolusi rendah *low resolution* (LR). Generator GAN meng-*upsampling* gambar LR ke gambar *super-resolution* (SR). Diskriminator digunakan untuk membedakan gambar HR dan *backpropagate loss* GAN untuk melatih diskriminator dan generator [27] seperti yang ditampilkan pada gambar 2.10.



Gambar 2.10 Proses training SRGAN

Pada gambar 2.10, jaringan untuk *generator* dan *discriminator* sebagian besar terdiri dari *convolutional layer*, *batch normalization* dan *parameterized* ReLU (PReLU). *Generator* juga mengimplementasikan koneksi loncat (*skip connection*) yang mirip dengan ResNet. *Convolutional layer* dengan "k3n64s1" adalah singkatan dari *3x3 kernel filter* yang menghasilkan 64 *channel* (saluran) dengan stride 1.



Gambar 2.11 Arsitektur SRGAN

SRGAN menggunakan *perceptual loss function* (L_{SR}) yang merupakan penjumlahan *weight* dari dua *loss component*, yaitu: *content loss* dan *adversarial loss*. *Loss* ini sangat penting untuk kinerja arsitektur generator [28].

2.8 Permasalahan dalam Deep Learning

Terdapat beberapa permasalahan yang sering dihadapi ketika mengimplementasi *deep learning*, terutama dalam *domain computer vision*. Di bawah merupakan salah satu permasalahan umum yang sering dihadapi para peneliti:

2.8.1 Overfitting

Overfitting terjadi saat model menjadi terlalu kompleks atau saat model dilatih dalam waktu yang berlebihan. Akibatnya, model mulai menyerap detail dan noise yang ada dalam data pelatihan melebihi tingkat yang seharusnya. Hasilnya, model menunjukkan kinerja yang sangat baik saat diuji dengan data pelatihan, tetapi kinerja yang buruk saat dihadapkan pada data yang belum pernah ditemui sebelumnya, seperti data validasi atau pengujian. Dengan kata lain, model kehilangan kemampuan untuk menggeneralisasi dengan efektif dari data pelatihan ke data baru [29].

2.8.2 Underfitting

Underfitting terjadi saat model memiliki kesederhanaan yang berlebihan untuk menangkap seluruh struktur dan pola yang terkandung dalam data. Model yang mengalami underfitting mungkin memiliki keterbatasan kapasitas dalam memahami data, atau mungkin tidak mendapatkan pelatihan yang mencukupi untuk memahami pola yang ada. Akibatnya, hasil kinerja model akan jelek tidak hanya pada data pelatihan, tetapi juga pada data validasi ataupun pengujian [30].

2.8.3 Adversarial Attacks

Adversarial attacks merupakan istilah di mana model *deep learning* dapat rentan terhadap serangan yang dimaksudkan untuk memperkenalkan gangguan kecil pada data input sehingga model menghasilkan prediksi yang salah. Ini merupakan masalah serius dalam penerapan di dunia nyata.

2.8.4 Interpretabilitas

Banyak model *deep learning*, terutama model yang sangat kompleks seperti CNN dan *Transformer*, sulit diinterpretasikan oleh manusia. Ini menjadi masalah dalam skenario di mana diperlukan pemahaman tentang mengapa dan bagaimana model membuat prediksi tertentu.

2.8.5 Permasalahan Domain Khusus

Beberapa masalah dalam *computer vision* terjadi dalam domain khusus, seperti deteksi medis atau pengenalan wajah, yang memerlukan pendekatan yang cermat dan sering kali data yang spesifik.

2.8.6 Keterbatasan Perangkat Keras dan Perangkat Lunak

Keterbatasan perangkat keras dan perangkat lunak dapat membatasi efisiensi dan performa model *deep learning*.

2.9 Loss Function

Loss Function (Fungsi Kerugian) merupakan komponen krusial dalam proses pembelajaran mesin dan pembelajaran mendalam. Fungsi ini bertindak sebagai

metrik yang mengukur sejauh mana prediksi model berbeda dari nilai sebenarnya. Dengan kata lain, fungsi kerugian memberikan ukuran tentang seberapa baik atau buruk performa model dalam memprediksi hasil yang diinginkan.

Pemilihan fungsi kerugian yang tepat memiliki dampak signifikan terhadap cara model mempelajari data. Fungsi kerugian yang tidak tepat dapat menghasilkan model yang tidak optimal atau bahkan gagal dalam belajar. Di sisi lain, pemilihan fungsi kerugian yang sesuai akan mengarahkan proses optimalisasi model secara efisien, membantu model memahami pola yang tepat dari data, dan menghasilkan prediksi yang akurat. Berikut ini adalah beberapa contoh fungsi kerugian penting yang umumnya digunakan oleh penulis:

2.9.1 Perceptual Loss

Perceptual Loss, yang juga dikenal sebagai loss konten, merupakan varian fungsi kerugian yang diaplikasikan dalam beberapa skenario seperti super-resolusi. Fungsi kerugian ini mengevaluasi disparitas di antara fitur-fitur berperingkat tinggi yang ditarik keluar dari input dan tujuan oleh jaringan saraf yang telah dipersiapkan terlebih dahulu [31].

Perceptual loss merupakan pendekatan yang lebih tingkat tinggi dalam mengukur perbedaan antara gambar yang dihasilkan oleh model dengan gambar asli. Fungsi kerugian ini berfokus pada kesamaan fitur atau representasi visual antara gambar yang dihasilkan dan gambar referensi (asli) menggunakan jaringan saraf tiruan yang telah dilatih sebelumnya (*pre-trained*), seperti jaringan konvolusi seperti VGG atau ResNet.

Dengan menggunakan representasi fitur ini, perceptual loss berupaya untuk memastikan bahwa gambar yang dihasilkan memiliki struktur dan pola yang mirip dengan gambar asli dalam hal representasi visual. Pendekatan ini membantu mengatasi masalah ketika metrik tradisional seperti *Mean Squared Error* (MSE) tidak mampu mengukur perbedaan gambar yang memiliki kualitas visual yang sama tetapi memiliki perbedaan dalam piksel yang kecil.

Pada perancangan sistem ini, digunakan *Adversarial Loss* yaitu BCE *Loss* yang merupakan *binary classification* untuk keluaran dari *discriminator*. Selain itu, *perceptual loss function* yang digunakan adalah VGG *Loss*.

Perceptual loss biasanya diukur menggunakan *MSE* antara fitur tingkat tinggi yang diekstraksi dari input dan target. *Misalkan F(x)* dan *F(y)* adalah fitur yang diekstraksi dari input *x* dan target *y*, maka *perceptual loss* didefinisikan sebagai:

$$L(x, y) = \frac{1}{n}\sum_{i=1}^{n}(F(x)_i - F(y)_i)^2$$

(2. 2)

Di mana:

x = Input

y = Target

$F(x)$ = Hasil ekstrasi input

$F(y)$ = Hasil ekstrasi output

2.9.2 Adversarial Loss

Adversarial loss merupakan konsep yang mendasari jaringan GAN. GAN adalah arsitektur yang terdiri dari dua komponen utama: *generator* (pembuat) dan *discriminator* (penilai). *Adversarial loss* berfokus pada interaksi antara *generator* dan *discriminator*.

Generator berusaha untuk menghasilkan data yang seolah-olah berasal dari distribusi data asli, sementara *discriminator* berusaha membedakan antara data asli dan data yang dihasilkan oleh *generator*. Proses ini mirip dengan permainan "*adversarial*", di mana *generator* berupaya untuk "menipu" *discriminator*, dan *discriminator* berusaha untuk menjadi semakin baik dalam mengenali data palsu.

Adversarial loss dihasilkan dari perbedaan antara probabilitas bahwa *discriminator* memprediksi data palsu berasal dari *generator* dan probabilitas

bahwa *discriminator* memprediksi data asli. Tujuan akhirnya adalah mengajari *generator* untuk menghasilkan data yang sangat mirip dengan data asli sehingga *discriminator* kesulitan membedakannya.

Persamaan 2.3 di bawah ini merupakan rumus perhitungan *adversarial loss*. Yang mana fungsi *generator* sebagai G dan fungsi *discriminator* sebagai D. Output *discriminator* berkisar sekitar 0 hingga 1, yang mewakili probabilitas bahwa gambar input adalah nyata.

$$Min_G\, Max_D\, V(D,G) = E_x[log\, D(x)] + E_z[log(1 - D(G(z)))]$$

(2. 3)

Di mana:

G = *Generator*

D = *Discriminator*

$D(x)$ = *Discrminator* menganggap data asli (x)

$G(z)$ = Data palsu (z) yang dihasilkan generator

2.10 Evaluasi Model

Penilaian model melibatkan proses mengevaluasi kinerja model *machine learning* atau *deep learning*. Tujuannya adalah untuk mengukur seberapa baik model mampu memberikan prediksi yang akurat dan dapat diandalkan berdasarkan data yang belum pernah dilihat sebelumnya. Proses evaluasi model memiliki signifikansi penting dalam pengembangan model, karena membantu menilai sejauh mana model telah berhasil memahami pola-pola yang terdapat dalam data dan sejauh mana model tersebut mampu melakukan generalisasi pada data baru.

2.10.1 Evaluasi Model Secara Umum

Dalam konteks *machine learning* dan *deep learning*, penilaian model sering melibatkan pemanfaatan ukuran-ukuran seperti akurasi, presisi, *recall*, dan *F1-score*. Parameter-parameter ini memberikan ikhtisar tentang performa model dalam

berbagai dimensi, dan dipilih sesuai dengan karakteristik serta kebutuhan khusus dari permasalahan yang dihadapi.

Terdapat beberapa metrik yang umum digunakan dalam evaluasi model dalam konteks *machine learning* dan *deep learning*. Berikut penjelasan singkat tentang beberapa di antaranya:

- **Akurasi (*Accuracy*)**: Akurasi adalah ukuran sejauh mana model mampu mengklasifikasikan data dengan benar. Ini adalah rasio antara jumlah prediksi yang benar dan total jumlah prediksi.

$$Accuracy = \frac{True\ Positives + True\ Negatives}{Total\ Prediksi}$$

(2. 4)

- **Presisi (*Precision*)**: Presisi mengukur seberapa akurat model dalam mengidentifikasi positif sejati dari semua prediksi positif. Ini dihitung sebagai rasio antara jumlah benar positif (*True Positive*) dan jumlah semua prediksi positif (*True Positive + False Positive*).

$$Precision = \frac{True\ Positives}{True\ Positives + False\ Positives}$$

(2. 5)

- ***Recall***: *Recall*, juga dikenal sebagai Sensitivitas (*sensitivity*) atau *True Positive Rate*, mengukur seberapa baik model dapat mengidentifikasi semua contoh positif yang sebenarnya. Ini dihitung sebagai rasio antara jumlah benar positif (*True Positive*) dan jumlah semua contoh positif sejati (*True Positive + False Negative*).

$$Recall = \frac{True\ Positives}{True\ Positives + False\ Negatives}$$

(2. 6)

- ***F1-Score***: *F1-score* adalah rata-rata harmonik antara presisi dan *recall*. Ini memberikan keseimbangan antara presisi dan *recall*, yang bermanfaat ketika kelas positif dan negatif memiliki distribusi yang tidak seimbang.

$$F1 - Score = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

(2. 7)

- ***Mean Squared Error* (MSE)**: MSE mengukur rata-rata dari kuadrat selisih antara prediksi dan nilai aktual. Ini juga umum digunakan dalam masalah regresi.

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

(2. 8)

Di mana:

N = Jumlah total sample

$\hat{y}_i$ = Nilai yang diprediksi oleh model untuk sampel ke-i.

y_i = Nilai sebenarnya (*ground truth*) dari sampel ke-i.

- ***Mean Absolute Error* (MAE)**: MAE juga disebut dengan *L1 Loss*, yaitu untuk mengukur rata-rata dari selisih absolut antara prediksi model dan nilai aktual. Ini digunakan dalam masalah regresi.

$$MAE = \frac{\sum_{i=1}^{N}|y_i - x_i|}{N}$$

(2. 9)

Di mana:

N = Jumlah total sample

y_i = Nilai sebenarnya (*ground truth*) dari sampel ke-i

x_i = Nilai yang diprediksi oleh model untuk sampel ke-i

2.10.2 Evaluasi Model Khusus Perbaikan Gambar

Metrik evaluasi khusus dalam konteks perbaikan gambar, seperti PSNR (*Peak Signal-to-Noise Ratio*), SSIM (*Structural Similarity Index*), dan MOS (*Mean Opinion Score*), membantu untuk mengukur sejauh mana kualitas gambar hasil perbaikan mendekati gambar asli yang sebenarnya. Berikut penjelasan dari ketiga metrik evaluasi tersebut:

- **PSNR (*Peak Signal-to-Noise Ratio*)**

  PSNR adalah ukuran kuantitatif yang digunakan untuk mengukur sejauh mana gambar hasil perbaikan mendekati gambar asli dengan memperhitungkan perbedaan antara sinyal gambar dan tingkat derau (noise) dalam gambar tersebut. PSNR diukur dalam satuan desibel (dB) dan diperoleh dengan menghitung perbandingan antara nilai maksimal sinyal gambar asli dengan rata-rata kuadrat perbedaan antara piksel gambar asli dan piksel gambar hasil perbaikan [32]. Nilai ideal dari PSNR yaitu sekitar 30 dB (desibel), namun ada juga yang di bawah 30 db (tergantung gambar inputnya). Semakin tinggi nilai PSNR maka semakin tinggi tingkat kemiripan antara gambar asli dengan gambar hasil dari generator model (gambar buatan). Rumus matematis PSNR dinyatakan pada persamaan 2.10 di bawah.

$$PSNR(I_1, I_2) = 10 \log_{10} \left(\frac{R^2}{MSE(I_1, I_2)} \right) \quad (2.10)$$

Di mana:

I_1 = Citra asli atau referensi yang dianggap sebagai "*ground truth*".

I_2 = Citra yang telah direstorasi.

R = Rentang nilai piksel dalam citra.

Dari persamaan 2.10, maka untuk MSE menjadi:

$$MSE(I_1, I_2) = \frac{1}{MN}\sum_{i=1}^{M}\sum_{j=1}^{N}[I_1(i,j) - I_2(i,j)]^2$$

(2. 11)

- **SSIM (*Structural Similarity Index*)**

  SSIM adalah metrik yang lebih kompleks daripada PSNR dan lebih mencerminkan persepsi visual manusia terhadap gambar. SSIM memperhitungkan tiga aspek utama: perbandingan kontras, perbandingan struktur, dan perbandingan kecerahan antara gambar asli dan hasil perbaikan [32]. Rentang nilai dari SSIM yaitu dari -1 sampai 1. Di mana nilai yang mendekati 1 menunjukkan kualitas gambar yang tinggi dan hampir serupa dengan gambar aslinya. Rumus matematis SSIM dinyatakan pada persamaan 2.12 dan 2.13 di bawah.

$$SSIM(I_1, I_2) = l(I_1, I_2)c(I_1, I_2)s(I_1, I_2)$$

(2. 12)

Di mana:

$$l(I_1, I_2) = \frac{2\mu_{I_1}\mu_{I_2} + C_1}{\mu_{I_1}^2 + \mu_{I_1}^2 + C_1}$$

$$c(I_1, I_2) = \frac{2\sigma_{I_1}\sigma_{I_2} + C_2}{\sigma_{I_1}^2 + \sigma_{I_1}^2 + C_2}$$

$$s(I_1, I_2) = \frac{\sigma_{I_1 I_2} + C_3}{\sigma_{I_1}\sigma_{I_2} + C_3}$$

(2. 13)