BAB II
TINJAUAN PUSTAKA

2.1. Penelitian Sebelumnya

Dalam penelitian sebelumnya, terdapat beberapa jurnal mengenai pembuatan kamera CCTV otomatis. Tetapi, banyak penelitian yang gagal untuk mengimplementasi kamera otomatis berbasis identifikasi wajah dikarenakan kamera resolusi rendah. Jurnal [10] membahas secara khusus mengenai hal tersebut. Salah satu jurnal memberi saran untuk menggunakan *FaceNet* dan MTCNN dalam Jetson TX2 [11]. Tetapi, Jetson TX2 memiliki harga yang mahal.

Berdasarkan permasalahan yang terjadi, para peneliti mulai mengembangkan cara untuk meningkatkan resolusi gambar. Salah satu metode terdahulu adalah SRGAN (*Super Resolution Generative Adversarial Network*) [8]. SRGAN merupakan pendekatan inovatif yang diperkenalkan pada tahun 2016. Metodologi ini biasa digunakan sebagai acuan untuk metode-metode baru, terutama dalam pendekatan GAN. SRGAN dirancang untuk menghasilkan gambar beresolusi tinggi dari input resolusi rendah, dengan penekanan pada pemulihan detail halus yang sering terlewatkan oleh Teknik resolusi super berbasis jaringan saraf konvolusional tradisional.

Dikarenakan penelitian SRGAN, beberapa peneliti mencoba untuk mengimplementasikannya ke beberapa perangkat, salah satunya adalah kamera CCTV. Pada tahun 2021, penelitian yang berhubungan dengan kamera CCTV resolusi tinggi telah diusulkan dengan menggunakan SRGAN dan CLAHE dengan tujuan untuk mengembangkan gambar *super realistic* dan resolusi tinggi [12]. Hasil dari penelitian tersebut memperlihatkan bahwa SRGAN merupakan metode yang efektif untuk meningkatkan resolusi gambar. Tetapi, SRGAN tidak dapat memprediksi wajah seseorang pada gambar dengan piksels yang rusak (terlalu *blur* atau terguncang).

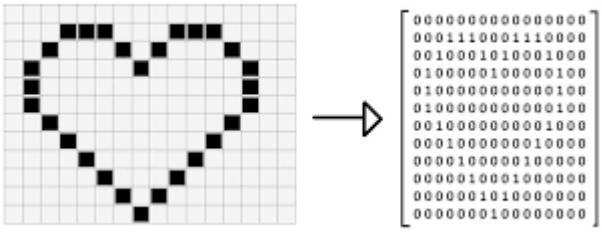
Penelitian dilanjutkan dengan metode baru yang biasa disebut dengan restorasi wajah (*face restoration*).

Telah banyak implementasi mengenai restorasi wajah. Jurnal [13] membahas metode apa saja yang telah dikembangkan hingga tahun 2022. *Gan Prior Embedded Network* (GPEN) [9] merupakan salah satu metode pertama yang menggunakan pendekatan GAN untuk melakukan restorasi wajah. GPEN mewakili terobosan untuk *blind face restoration* (BFR) dalam pengaturan yang tidak terkontrol. Arsitekturnya mengintegrasikan *pre-trained* GAN yang dilatih untuk pembuatan gambar wajah berkualitas tinggi dan decoder *Deep Neural Network*. Kombinasi ini memungkinkan GPEN untuk memulihkan gambar yang terdistorsi atau rusak parah, menetapkan tolok ukur baru untuk kinerja pemulihan wajah buta. Metodologi ini mendemonstrasikan kekuatan memanfaatkan pengetahuan sebelumnya dari GAN, yang secara efektif merangkum bermacam-macam gambar alami, untuk memulihkan gambar wajah bahkan di bawah oklusi yang parah.

2.2. Pengolahan Citra Digital

Istilah pemrosesan gambar digital (*Image processing*) pada umumnya berkaitan dengan penanganan komputasi grafis dua dimensi. Konsep ini juga dapat menyiratkan penanganan digital data dua dimensi apa pun. Secara umum, gambar digital adalah susunan bilangan nyata atau bilangan kompleks yang diekspresikan melalui sejumlah bit yang terbatas [14].

Ketika merepresentasikan gambar, kita harus memberikan perhatian khusus pada atribut yang dapat diukur dari setiap elemen gambar, yang sering dikenal sebagai piksel (*pixel*). Gambar digital dapat ditransformasikan melalui matriks karena gambar dapat dipecah menjadi segmen bit atau piksel. Gambar 2.1 merupakan contoh pemrosesan gambar terkomputerisasi.



Gambar 2. 1 Representasi Citra Dalam Matriks

Dapat dilihat dari gambar 2.1 setiap fungsi dua dimensi yang berisi informasi dapat dilihat sebagai gambar. Sebuah model gambar menawarkan garis besar logis atau kuantitatif dari properti yang ada dalam fungsi ini. Namun, dalam ranah gambar digital standar, nilai dalam setiap titik matriks sangat kompleks dan kaya akan informasi. Hal ini menyiratkan bahwa gambar digital pada umumnya menampilkan matriks tiga dimensi di mana setiap dimensi menandakan warna merah, hijau, dan biru.

2.2.1. Filtering

Filtering pada citra digital adalah operasi utama dalam bidang pemrosesan gambar dan *computer vision*, yang berhubungan dengan modifikasi atau peningkatan gambar. Prosesnya melibatkan melakukan operasi matematika tertentu pada nilai piksel gambar. Operasi ini biasanya dijalankan melalui konvolusi antara gambar dan matriks yang telah ditentukan sebelumnya, sering disebut dengan *kernel* atau *filter*.

Setiap *kernel* menyelesaikan tugas yang berbeda-beda. Jenis *kernel* yang digunakan menentukan sifat operasi yang dilakukan pada gambar. Berikut ini adalah contoh dari beberapa jenis *kernel* yang umum digunakan:

- ***Box (Mean) Filter*:** *Box filter* menggantikan nilai setiap piksel dengan nilai rata-rata piksel tetangganya, yang mencakup piksel itu sendiri. *Box filter* terutama digunakan untuk mengaburkan gambar dan mengurangi kebisingan. Ukuran kotak menentukan tingkat kekaburan.

- ***Gaussian Filter*:** *Gaussian filter* merupakan representasi dari *low-pass filter*, yang berfungsi untuk mengatasi interferensi berfrekuensi tinggi dalam sebuah gambar. Mekanisme *Gaussian filter* beroperasi dengan memberikan bobot lebih signifikan pada pixel yang berlokasi di pusat jendela, sementara secara simultan menurunkan bobot pada piksel yang terletak di bagian tepian, secara paralel dengan prinsip dari distribusi *gaussian*.
- ***Laplacian Filter*:** *Laplasian filter* digunakan untuk deteksi tepi pada gambar. *Filter* ini menghitung turunan kedua dari suatu gambar, menandai wilayah di mana perubahan intensitas yang cepat terjadi. Tidak seperti *filter* lain yang mengaburkan gambar atau mengurangi kebisingan, *laplasian filter* mempertajam gambar.
- ***Sobel Filter*:** *Sobel filter* juga digunakan untuk deteksi tepi. *Filter* ini menghitung turunan pertama dari suatu gambar, yang memberikan informasi tentang tepi dan gradien gambar. Ada dua *filter* sobel, satu untuk perubahan horizontal, dan satu lagi untuk vertikal.
- ***Scharr Filter*:** *Scharr filter* adalah versi perbaikan dari *sobel filter*, memberikan deteksi tepi yang lebih akurat. *Filter* ini menyelesaikan hal tersebut dengan menggunakan kernel konvolusi yang lebih besar, yang memperhitungkan wilayah yang lebih besar di sekitar setiap piksel.

2.2.2. Convolution

Konvolusi (*convolution*) merupakan operasi matematika kunci yang memegang peran penting dalam ranah pemrosesan citra. Sebagai kerangka kerja dari sejumlah besar prosedur yang bertujuan untuk memanipulasi atau mengubah gambar, konvolusi berfungsi dengan cara menggerakkan sebuah matriks - yang biasanya kita kenal sebagai kernel atau filter - di sepanjang gambar yang kita masukkan. Kemudian, konvolusi mengkalkulasi hasil perkalian dari setiap lokasi untuk menghasilkan gambar keluaran. Melalui operasi ini, kita dapat melakukan berbagai jenis perubahan pada gambar, seperti pengaburan, penajaman, hingga deteksi tepi.

Ukuran, bentuk, dan koefisien *kernel* menentukan sifat dan jangkauan operasi yang dilakukan konvolusi. Intinya, setiap piksel dalam gambar adalah jumlah tertimbang dari piksel masukan yang sesuai dan tetangganya. Bobot ditentukan oleh *kernel*, dan penerapannya secara inheren membentuk karakter pemrosesan. Konvolusi dapat ditulis sebagai berikut:

$$(\boldsymbol{I} \cdot \boldsymbol{K})[p, q] = \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} (\boldsymbol{I}[p + i, q + j] \cdot \boldsymbol{K}[i, j])$$

(2. 1)

Di mana:

$(\cdot)$ = denotasi operasi konvolusi.

p, q = Koordinat di dalam *output* gambar.

i, j = Koordinat di dalam *kernel* K.

$\sum_{i=0}^{m-1} \sum_{j=0}^{n-1}$ = Mewakili jumlah dari dimensi *kernel*.

2.2.3. Segmentasi

Segmentasi citra secara formal adalah proses pemisahan suatu citra digital menjadi beberapa segmen (set piksel). Dengan kata lain, kita bisa memecah suatu citra menjadi beberapa bagian, yang dapat membuat citra lebih mudah dianalisis dan dipahami. Anggap saja kita sedang mengurai teka-teki kompleks menjadi potongan-potongan yang lebih kecil dan lebih mudah untuk ditangani. Segmen-segmen ini bukan hanya sekumpulan piksel acak, melainkan juga membentuk representasi yang lebih bermakna dan lebih mudah untuk dianalisis dibandingkan dengan representasi visual citra secara keseluruhan [15][16].

Secara umum, segmen-segmen dapat dianggap sebagai petunjuk dalam mencari batas dan lokasi objek dalam suatu citra. Mereka berfungsi sebagai

penunjuk penting dalam banyak aplikasi pengolahan citra, serupa dengan petunjuk yang membantu kita menyelesaikan teka-teki, yang meliputi analisis citra medis, pemahaman adegan, pengenalan pola, hingga pengawasan video.

Seiring berjalannya waktu, metode yang berbasis pada *deep learning* (DL) telah menjadi semacam 'panduan ahli' dalam dunia segmentasi citra. Dengan bantuan model-model seperti U-Net dan variasinya, kita mendapatkan alat yang sangat efektif, layaknya memiliki seorang ahli teka-teki yang membantu kita menyelesaikan teka-teki yang rumit [17][18]. Metode ini melibatkan penggunaan jaringan saraf konvolusional untuk memahami fitur-fitur penting dari citra, mirip dengan bagaimana kita memahami dan mengenali potongan-potongan teka-teki yang penting, dan kemudian menggunakan fitur-fitur tersebut untuk melakukan proses segmentasi. Layaknya menyelesaikan teka-teki, kita mengambil segmen-segmen citra, belajar dari mereka, dan menyusun mereka kembali menjadi suatu gambaran yang lebih bermakna dan lebih mudah untuk kita pahami.

2.3. *Machine Learning*

Machine Learning (ML) telah menjadi teknologi inti dalam Revolusi Industri Keempat (4IR atau Industri 4.0) saat ini karena kemampuannya untuk belajar dari data. *Deep Learning* (DL), yang merupakan cabang dari ML telah muncul sebagai pendekatan populer di berbagai bidang aplikasi seperti perawatan kesehatan, pengenalan visual, analisis teks, *cyber security*, dan lain-lain. Namun, membangun model DL yang tepat adalah tugas yang menantang karena sifat dinamis dan variasi dalam masalah dan data dunia nyata [19].

Teknologi DL, yang berasal dari jaringan syaraf tiruan (*artificial neural network*/ANN), sering dianggap sebagai *black box* karena kurangnya pemahaman inti. Hal ini menghambat pengembangan pada tingkat standar. Oleh karena itu, pandangan yang terstruktur dan komprehensif tentang teknik DL, termasuk taksonomi yang mempertimbangkan berbagai jenis tugas dunia nyata seperti pembelajaran yang diawasi atau tidak diawasi, sangat penting. Dalam taksonomi ini, jaringan yang dalam untuk pembelajaran yang diawasi atau diskriminatif,

pembelajaran yang tidak diawasi atau generatif, serta pembelajaran hibrida dan yang lainnya yang relevan diperhitungkan.

Salah satu tantangan utama dalam DL adalah kebutuhan data berlabel berskala besar untuk melatih jaringan syaraf tiruan. Hal ini sangat penting untuk mencapai kinerja yang lebih baik dalam pembelajaran fitur visual dari gambar atau video untuk aplikasi *computer vision*. Namun, biaya yang besar untuk mengumpulkan dan membuat anotasi set data berskala besar telah menyebabkan munculnya metode pembelajaran yang diawasi sendiri. Metode-metode ini, bagian dari metode pembelajaran tanpa pengawasan, diusulkan untuk mempelajari fitur gambar dan video umum dari data berskala besar yang tidak berlabel tanpa menggunakan label yang dianotasi manusia [20].

Metode *self-supervised learning* telah ditinjau secara ekstensif, terutama yang didasarkan pada pembelajaran mendalam. Motivasi, alur umum, dan terminologi bidang ini telah dijelaskan, dan arsitektur jaringan saraf dalam yang umum digunakan untuk pembelajaran yang diawasi sendiri telah dirangkum. Skema dan metrik evaluasi dari metode pembelajaran yang diawasi sendiri juga telah ditinjau, bersama dengan kumpulan data yang umum digunakan untuk gambar, video, audio, dan data 3D. Perbandingan kinerja kuantitatif dari metode yang ditinjau pada dataset benchmark telah dirangkum dan didiskusikan untuk pembelajaran fitur gambar dan video.

Di sisi penginderaan jauh, model DL telah digunakan untuk mengidentifikasi tajuk pohon pada citra RGB dengan menggunakan *semi-supervised deep learning detection network*. Meskipun terdapat keterbatasan dalam pendekatan deteksi tanpa pengawasan, data pelatihan yang berisik ini dapat mengandung informasi yang dapat digunakan jaringan saraf untuk mempelajari fitur-fitur pohon awal. Model awal kemudian disempurnakan dengan menggunakan sejumlah kecil gambar RGB beranotasi tangan yang berkualitas lebih tinggi. Pendekatan pembelajaran mendalam semi-pengawasan ini menunjukkan bahwa penginderaan jauh dapat mengatasi kurangnya data pelatihan berlabel dengan menghasilkan data berisik

untuk pelatihan awal menggunakan metode tanpa pengawasan dan melatih ulang model yang dihasilkan dengan data berlabel berkualitas tinggi [21].

Self-supervised learning (SSL) telah mencapai kinerja yang menjanjikan pada tugas-tugas pembelajaran bahasa alami dan gambar. Baru-baru ini, ada kecenderungan untuk memperluas keberhasilan tersebut ke data grafik menggunakan jaringan saraf tiruan (GNN). Metode SSL untuk GNN dapat dikategorikan ke dalam model kontras dan prediktif. Dalam kedua kategori tersebut, kerangka kerja terpadu untuk metode disediakan, bersama dengan bagaimana metode-metode ini berbeda dalam setiap komponen di bawah kerangka kerja. Perlakuan terpadu dari metode SSL untuk GNN ini menjelaskan kesamaan dan perbedaan berbagai metode, menyiapkan panggung untuk mengembangkan metode dan algoritma baru [22].

Kesimpulannya, ML, dan khususnya DL, telah menjadi teknologi utama di era digital saat ini. Namun, kebutuhan akan data berlabel berskala besar menghadirkan tantangan yang signifikan. Metode *self-supervised learning* menawarkan solusi yang menjanjikan untuk masalah ini, memungkinkan pembelajaran fitur umum dari data berskala besar yang tidak berlabel. Metode-metode ini telah diterapkan di berbagai bidang, termasuk visi komputer dan penginderaan jarak jauh, dan sekarang diperluas ke data grafik menggunakan jaringan syaraf tiruan.

2.4. *Deep Learning*

Untuk memecahkan hal yang kompleks, para peneliti mulai mengembangkan *deep learning* (DL). DL merupakan subbidang dari ML yang khusus digunakan dalam data yang banyak. Meskipun DL dapat digunakan dalam data yang sedikit, tetapi akan lebih efisien menggunakan algoritma ML tradisional. Yann LeCun dan berbagai peneliti lainnya membahas pada jurnal [23] bahwasanya DL cocok digunakan untuk *big data*. Teknis ML tradisional sering kesulitan untuk menangkap pola dan ketergantungan kompleks dalam kumpulan *big data* karena biasanya mengandalkan fitur rekayasa tangan dan arsitektur dangkal. DL, di sisi lain, unggul

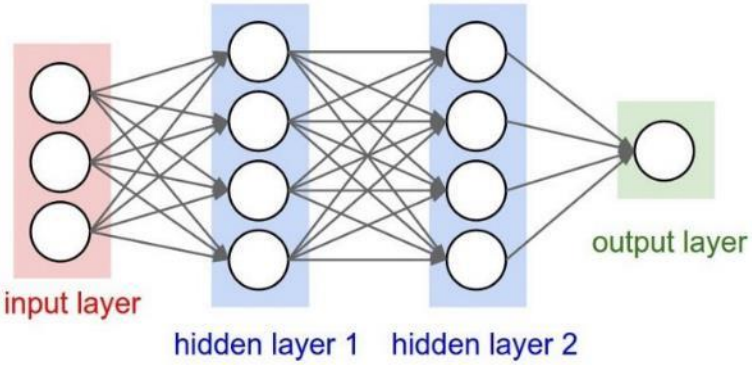
dalam skenario ini karena menggunakan banyak lapisan jaringan syaraf tiruan untuk mempelajari representasi fitur hierarkis langsung dari data, memungkinkannya mengekstraksi fitur berguna secara otomatis dan menangani seluk-beluk *big data*. Terdapat tiga algoritma utama *deep learning*, yaitu: *artificial neural network* (ANN), *convolutional neural network* (CNN), dan *recurrent neural network* (RNN). Pada skripsi yang ditulis, penulis berfokus ke dalam algoritma CNN, meskipun akan ada beberapa perbandingan dengan model yang menggunakan *transformer*, yang merupakan versi peningkatan untuk RNN.

2.4.1. Artificial Neural Network

Artificial neural network (ANN) adalah sistem komputasi yang dirancang berdasarkan prinsip kerja jaringan saraf biologis yang ada di otak. Sistem ini memiliki kemampuan untuk belajar melalui sejumlah contoh tanpa perlu didefinisikan aturan-aturan khusus yang bersifat deterministik. Sebagai ilustrasi, ketika ditugaskan untuk membedakan antara gambar anjing dan kucing, sistem ini tidak memerlukan penulisan aturan-aturan spesifik oleh programmer. Sebaliknya, dengan memasukkan sejumlah besar data berupa gambar kucing dan anjing serta menginformasikan sistem tentang klasifikasi tiap gambar tersebut, sistem akan memproses informasi ini dan "belajar" untuk membedakan antara gambar kucing dan anjing.

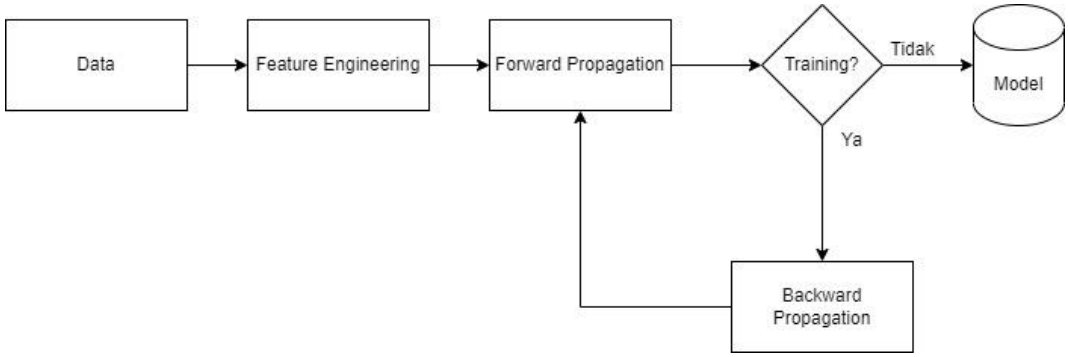
ANN tersusun dari berbagai lapisan yang mencakup *input layer*, *hidden layer*, dan *output layer*. Lapisan input adalah titik awal dari arsitektur ANN, tempat di mana data awal dimasukkan ke dalam sistem. Dari lapisan input, data akan diumpan ke dalam satu atau lebih *hidden layer*. *Hidden layer* berperan sebagai pusat pemrosesan utama dalam ANN, tempat di mana aktivasi dan penyesuaian bobot terjadi berdasarkan algoritma belajar yang dipilih. Setelah melalui proses di *hidden layer*, data akan diumpan ke lapisan *output*. *Output layer* merupakan titik akhir dari arsitektur ANN, tempat di mana hasil akhir, berdasarkan pemrosesan yang telah dilakukan oleh *hidden layer*, disajikan. Seluruh proses ini mengembangkan struktur interkonektivitas mirip otak yang memberikan ANN kemampuan belajar dan

generalisasi yang mendalam. Gambar 2.2 [24] merupakan ilustrasi dari arsitektur ANN.



Gambar 2. 2 Ilustrasi Arsitektur ANN

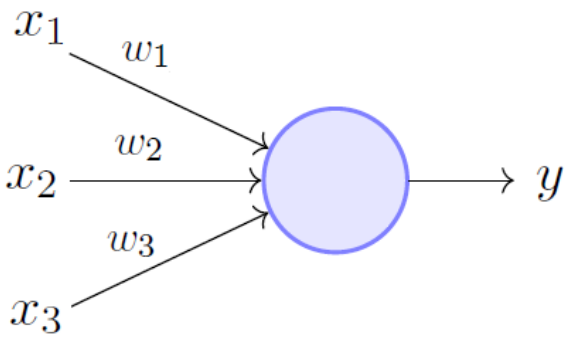
Gambar 2.2 menunjukkan salah satu contoh arsitektur ANN yang memiliki tiga input, dua *hidden layer*, dan satu *output*. Sebelum membahas secara detail mengenai mekanisme ANN, penting untuk memiliki pemahaman konseptual yang lebih luas tentang proses kerja ANN. Diagram pada gambar 2.3 memberikan gambaran umum tentang alur kerja ANN.



Gambar 2. 3 Diagram Alur Kerja ANN

Dalam gambar 2.3, dalam penerapan ANN, tahap pertama yang dilakukan adalah pengumpulan data. Data tersebut kemudian dibersihkan dan dipilih fitur-fitur yang relevan untuk digunakan dalam proses pelatihan model. Setelah data siap, sistem ANN akan melakukan proses yang dikenal sebagai *forward propagation*, di mana data input (X) diproses untuk memprediksi *output* (y). Untuk meningkatkan

kecerdasan model, dilakukan proses *backward propagation* dan proses ini diulangi hingga model mencapai tingkat efektivitas yang diinginkan. Untuk melihat ilustrasi lebih detail, gambar 2.4 adalah ilustrasi bagaimana setiap bulatan (*node*) menghasilkan nilai *output* (y) dari input (x). Proses ini biasa disebut dengan *perceptron*.



Gambar 2. 4 Ilustrasi Perceptron

Berdasarkan gambar 2.4, *perceptron* adalah algoritma dalam ANN yang digunakan untuk *supervised learning* dari pengklasifikasi biner, prediksi nilai kontinyu, dan aplikasi lainnya. *Perceptron* memproses input sebagai vektor dan mengarahkannya ke neuron berikutnya. ANN sejatinya merupakan kumpulan *perceptron* di mana setiap input diteruskan melalui lapisan tersembunyi (*hidden layer*) hingga menghasilkan *output*. ANN dapat dianggap sebagai *multi-layer perceptron* yang juga dikenal sebagai *feed-forward neural network*. Hal ini akan berisi:

- *Hidden layer*
- *Bias units*
- *Neurons* (input, *output*, dan *perceptron*)
- Bobot
- Fungsi aktivasi

Sebagai ilustrasi, misalkan kita mempunyai variabel input x_1, x_2, dan x_3 dan *output* $\hat{y}$. Tujuan dari ANN adalah bagaimana caranya mesin dapat memprediksi

output $\hat{y}$ dengan input yang diberikan. Untuk mencapai tujuan itu, kita inisialisasi bobot w_1, w_2, dan w_3. *Output* ŷ dapat diprediksi dengan persamaan:

$$\hat{y} = \sum_{i=1}^{N} x_i w_i + b$$

(2. 2)

Di mana:

$\hat{y}$ = *Output*.

x_i = input ke-i.

w_i = bobot ke-i.

b_i = *bias* ke-i.

Apabila rumusan di atas dijabarkan lebih lanjut, maka dapat dituliskan kembali sebagai berikut:

$$\hat{y} = x_1 w_1 + x_2 w_2 + x_3 w_3 + b$$

(2. 3)

Untuk membuat komputasi lebih cepat, kita dapat memasukkan input dan bobot ke dalam matriks dan menghitung perkalian titik (*dot product*) dengan persamaan:

$$\hat{Y} = X \cdot W + b$$

(2. 4)

Di mana:

X = input matriks.

W = bobot matriks.

b = *bias* vektor.

Dalam konteks jaringan saraf tiruan, *forward propagation* sering kali diringkas menjadi operasi matematika yang melibatkan dot product antara vektor masukan X dan matriks bobot W, ditambah dengan vektor bias b, yang umumnya direpresentasikan sebagai $Y = X \cdot W + b$. Walaupun rumus ini telah mencakup semua komponen esensial, ada pendekatan alternatif yang sering digunakan untuk mengintegrasikan bias ke dalam operasi *dot product* itu sendiri. Teknik ini tidak hanya mempermudah notasi tetapi juga memiliki keuntungan dalam efisiensi komputasional.

Pertama-tama, vektor masukan X yang berdimensi n dimodifikasi dengan menambahkan elemen bernilai 1, menghasilkan vektor baru X' menjadi $[1, x_1, x_2, \dots, x_n]$. Sementara itu, matriks bobot W juga mengalami modifikasi dengan penambahan baris yang memuat komponen bias w_0, sehingga dimensinya berubah dari $n \times m$ menjadi $(n+1) \times m$. Dengan melakukan transformasi ini, ekspresi dapat dereformulasi menjadi:

$$f(x) = (W^T X + w_0)$$

(2. 5)

Di mana:

$$W = \underset{m \times n}{\begin{bmatrix} w_{11} & \cdots & w_{1n} \\ \vdots & \ddots & \vdots \\ w_{m1} & \cdots & w_{mn} \end{bmatrix}} \; \& \; X = \underset{m \times k}{\begin{bmatrix} x_{11} & \cdots & x_{1k} \\ \vdots & \ddots & \vdots \\ x_{m1} & \cdots & x_{mk} \end{bmatrix}}$$

2.4.1.1. Fungsi Aktivasi

Secara umum, nilai yang dikeluarkan oleh sebuah *perceptron* akan diteruskan ke fungsi aktivasi sebelum diproses lebih lanjut. Fungsi aktivasi berfungsi sebagai komponen penting dalam jaringan saraf tiruan, memungkinkan model untuk memecahkan masalah *non-linear* yang kompleks dengan memasukkan properti *non-linear* ke dalam komputasi jaringan. Secara dominan, fungsi tersebut

diterapkan pada input berbobot setiap *neuron* untuk menentukan *output* dari *neuron* yang akan diteruskan ke lapisan berikutnya. Dengan demikian, fungsi aktivasi memainkan peran penting dalam mendefinisikan dan mengoptimalkan kinerja model jaringan saraf.

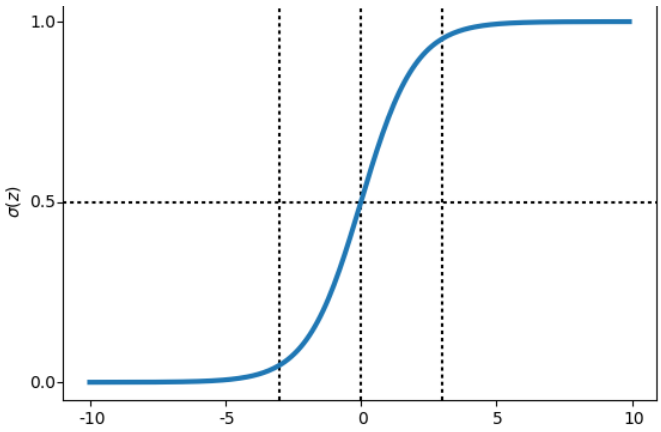
Ada beragam fungsi aktivasi, masing-masing dengan properti unik yang melayani berbagai jenis masalah. Misalnya, fungsi *sigmoid* digunakan secara luas karena kurva berbentuk S-nya, yang secara efektif menormalkan keluaran antara 0 dan 1. Fungsi *sigmoid* dapat ditulis dalam rumus:

$$\sigma(X) = \frac{1}{1 + e^{-X}} \tag{2. 6}$$

Di mana:

X = input matriks.

Untuk ilustrasi yang lebih baik, dapat dilihat gambar 2.5, yang merupakan kurva dari fungsi *sigmoid*.



Gambar 2. 5 Plot Kurva Sigmoid

Dalam gambar 2.5, dapat dilihat fungsi *sigmoid*. Bentuknya menyerupai huruf "S" dan memiliki rentang antara 0 dan 1. Fungsi ini sangat berguna dalam situasi di mana kita perlu mengubah nilai input yang beragam menjadi probabilitas

yang dapat diinterpretasikan. Dalam konteks *machine learning*, fungsi *sigmoid* sering digunakan sebagai fungsi aktivasi dalam jaringan saraf tiruan, memungkinkan model untuk memahami dan mempelajari relasi non-linear antara fitur. Fungsi *sigmoid* juga digunakan dalam regresi logistik, di mana outputnya dapat diinterpretasikan sebagai probabilitas bahwa sampel tertentu termasuk dalam kelas tertentu.

Alternatifnya, fungsi *Rectified Linear Unit* (ReLU), merupakan pilihan populer lainnya, yang mengatasi masalah *vanishing gradient* dengan mengeluarkan input secara langsung jika positif; jika tidak, hasilnya nol. Fungsi ReLU dapat ditulis dalam rumus:

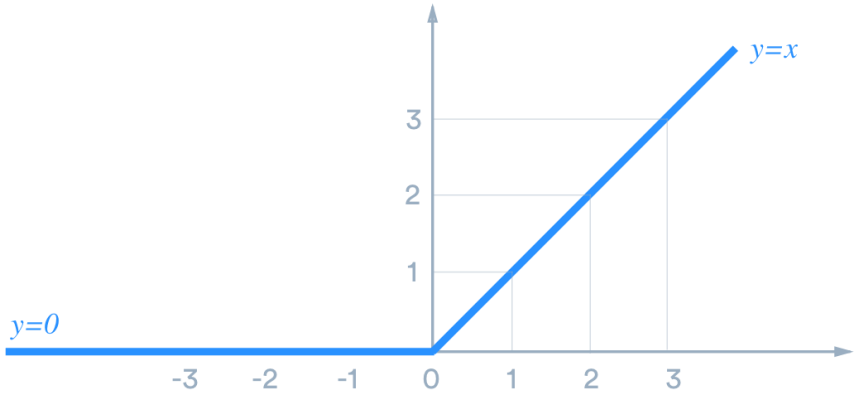
$$R(X) = \begin{cases} X \; if \; X > 0 \\ 0 \; otherwise \end{cases}$$

(2. 7)

Di mana:

X = input matriks.

Untuk ilustrasi yang lebih baik, dapat dilihat pada gambar 2.6, yang merupakan *plot* fungsi dari ReLU.



Gambar 2. 6 Plot Fungsi ReLU

Gambar 2.6 menunjukkan fungsi ReLU. Fungsi ReLU adalah fungsi aktivasi yang populer dalam *machine learning*. Fungsi ini mengubah semua input negatif menjadi 0 dan mempertahankan input positif seperti adanya.

Fungsi terakhir yang akan dibahas dalam penelitian ini adalah *Leaky Rectified Linear Unit* (*Leaky* ReLU). Pemilihan fungsi aktivasi ini dilakukan berdasarkan pertimbangan bahwa penelitian ini akan memanfaatkan fungsi aktivasi tersebut. *Leaky* ReLU merupakan variasi dari fungsi aktivasi ReLU; berbeda dengan ReLU yang memberikan nilai 0 untuk input negatif, *Leaky* ReLU mengalikan input negative dengan suatu konstanta θ. *Leaky* ReLU dapat ditulis dalam rumus:

$$LR(X) = \begin{cases} X \; if \; X > 0 \\ \theta X \; otherwise \end{cases}$$

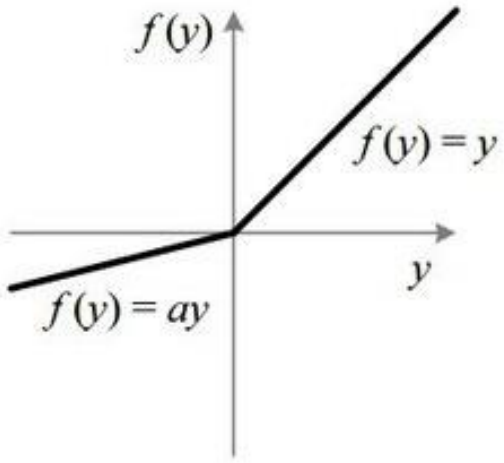
(2. 8)

Di mana:

X = input matriks.

θ = Skalar konstanta.

Untuk ilustrasi yang lebih baik, dapat dilihat pada gambar 2.7, yang merupakan *plot* fungsi *leaky* ReLU.



Gambar 2. 7 Plot Leaky ReLU

Gambar 2.7 menunjukkan fungsi *Leaky* ReLU. Fungsi *Leaky* ReLU adalah variasi dari ReLU yang memperbaiki kekurangannya. Fungsi ini memungkinkan nilai negatif bergerak ke arah nol, bukan langsung menjadi nol, sehingga mengurangi masalah neuron mati.

2.4.1.2. Feed-Forward dan Back-Propagation

Tujuan dari ANN adalah untuk melakukan estimasi terhadap fungsi f yang memberikan prediksi untuk sejumlah keluaran berdasarkan serangkaian masukan yang diberikan. ANN yang telah disebutkan sebelumnya dikenal sebagai umpan-maju (*feed-forward*) karena tidak melibatkan umpan balik (*back-propagation*) yang mengalir kembali ke lapisan input.

Dalam konteks ini, dapat dinyatakan bahwa hasil perhitungan pada ANN dapat diperoleh dengan mengalihkan input dengan bobot, menambahkan nilai *bias*, dan menerapkan fungsi aktivasi, yang dapat ditulis menjadi:

$$net_j = \sum_{i=1}^{d} x_i w_{ij} + w_{j0} = \sum_{i=0}^{d} x_i w_{ij}$$

$$\Rightarrow y_j = f(net_j)$$

(2. 9)

Di mana:

f = Fungsi aktivasi

Dalam setiap jaringan saraf, terdapat fungsi biaya (*cost function*) yang terkait dengan prediksinya. Fungsi biaya ini sering disebut sebagai fungsi kerugian (fungsi kerugian). Fungsi kerugian perlu dioptimalkan dengan memperbarui bobot yang ada. Untuk mencapai optimisasi tersebut, perhitungan turunan parsial fungsi

kerugian terhadap setiap bobot dapat dilakukan, dan bobot dapat diperbarui dengan mempertimbangkan fungsi kerugian yang optimal.

Ada berbagai jenis fungsi kerugian seperti *Mean Squared Error* (MSE), *Cross-Entropy*, *Log Loss*, dan *Hinge Loss*. Tetapi, fungsi kerugian tidak dibatasi pada jenis-jenis tersebut semata, karena dalam sistem yang kompleks, para peneliti seringkali merancang fungsi kerugian mereka sendiri. Masing-masing fungsi kerugian ini berlaku dalam situasi yang berbeda. Misalnya, MSE umumnya digunakan dalam tugas regresi, sedangkan *Cross-Entropy* sering digunakan dalam tugas klasifikasi. Pilihan fungsi kerugian tergantung pada masalah spesifik yang dihadapi.

Pada tahun 1986, Geoff Hinton mengusulkan metode propagasi balik dalam jurnalnya [25]. Pada saat itu, teori ini awalnya dianggap remeh, namun secara perlahan menjadi inti dari pengembangan jaringan saraf umpan-maju (*feed-forward neural networks*).

Metode propagasi balik memungkinkan perhitungan turunan parsial terhadap setiap bobot dengan cara mundur. Dengan demikian, dapat dinyatakan bahwa propagasi balik memungkinkan untuk menghitung turunan parsial untuk setiap bobot melalui proses mundur.

Ide inti dari propagasi balik adalah bahwa dengan mengetahui seberapa besar perubahan kecil pada bobot akan mempengaruhi keluaran, kita dapat menyesuaikan bobot untuk meminimalkan fungsi kerugian [26]. Hal ini dilakukan dengan menggunakan prinsip kalkulus, khususnya *chain rule*.

Propagasi balik dan fungsi kerugian terkait erat dalam proses pembelajaran jaringan saraf. Fungsi kerugian memberikan ukuran kesalahan keluaran jaringan saat ini, sedangkan propagasi balik menggunakan kesalahan ini untuk menyesuaikan bobot jaringan untuk meminimalkan kesalahan ini.

Sebagai contoh, mari pertimbangkan jaringan saraf yang dirancang untuk masalah klasifikasi biner. Fungsi kerugian yang digunakan dalam permasalahan ini

adalah *binary cross-entropy*, yang merupakan pilihan paling umum. Rumus dari *binary cross-entropy* adalah:

$$L_{BCE} = -\frac{1}{n}\sum_{i=1}^{n}(Y_i \cdot \log \hat{Y}_i + (1 - Y_i) \cdot \log(1 - \hat{Y}_i))$$

(2. 10)

Di mana:

n = Jumlah data.

Y_i = Target ke-i.

$\hat{Y}_i$ = Probabilitas yang diprediksi ke-i.

Setelah menetapkan fungsi kerugian, proses propagasi balik dapat dilaksanakan. Propagasi balik melibatkan empat langkah utama: *forward pass*, perhitungan fungsi kerugian, *backward pass*, dan pembaruan bobot.

Untuk ilustrasi, akan dipertimbangkan jaringan saraf dengan satu lapisan tersembunyi dan variabel berikut:

x = input.

w_1, w_2 = Bobot.

b_1, b_2 = *Biases*.

z_1, z_2 = Jumlah bobot masing-masing untuk lapisan tersembunyi dan lapisan keluaran.

h = Aktivasi lapisan tersembunyi.

$\hat{y}$ = Prediksi.

y = Label sebenarnya.

L = Fungsi kerugian.

f = fungsi aktivasi ReLU.

g = fungsi aktivasi *Sigmoid*.

Forward pass (dari input ke *output*) dapat dijelaskan dengan persamaan berikut:

$$z_1 = w_1 \cdot x + b_1$$

$$h = f(z_1)$$

$$z_2 = w_2 \cdot h + b_2$$

$$\hat{Y} = g(z_2)$$

(2. 11)

Sekarang, hitung fungsi kerugian menggunakan rumus *binary cross-entropy* seperti pada rumus 2.10.

Langkah selanjutnya adalah melaksanakan propagasi balik, di mana kita menghitung gradien kerugian untuk setiap bobot dan *bias* dengan menggunakan aturan *chain-rule differentiation*. Hal ini dapat diilustrasikan dengan persamaan berikut:

$$\frac{\partial L}{\partial z_2} = \hat{y} - y$$

$$\frac{\partial L}{\partial w_2} = \frac{\partial L}{\partial z_2} \cdot h$$

$$\frac{\partial L}{\partial b_2} = \frac{\partial L}{\partial z_2}$$

$$\frac{\partial L}{\partial h} = \frac{\partial L}{\partial z_2} \cdot w_2$$

$$\frac{\partial L}{\partial z_1} = \frac{\partial L}{\partial h} \cdot \frac{\partial}{\partial f(z_1)}$$

$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial z_1} \cdot x$$

$$\frac{\partial L}{\partial b_1} = \frac{\partial L}{\partial z_1}$$

(2. 12)

Terakhir, kita memperbarui bobot dan *bias* menggunakan penurunan gradien. Rumus untuk memperbarui bobot dan *bias* adalah:

$$w_1 = w_1 - \alpha \times \frac{\partial L}{\partial w_1}$$

$$b_1 = b_1 - \alpha \times \frac{\partial L}{\partial b_1}$$

$$w_2 = w_2 - \alpha \times \frac{\partial L}{\partial w_2}$$

$$b_2 = b_2 - \alpha \times \frac{\partial L}{\partial b_2}$$

(2. 13)

Di mana α merupakan *hyperparameter* konstanta yang menentukan ukuran langkah pada setiap iterasi sambil bergerak menuju fungsi kerugian minimum. Dalam praktiknya, seluruh proses (*forward pass*, perhitungan *loss*, propagasi balik, perbarui bobot) diulang beberapa kali pada data pelatihan, menyesuaikan bobot dan *bias* ke arah yang mengurangi *loss*, sehingga meningkatkan performa model.

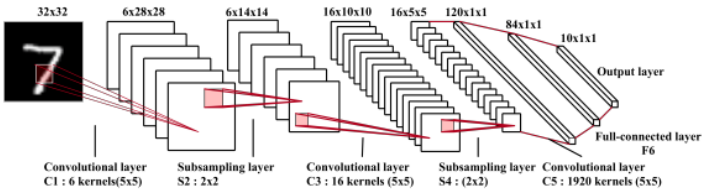
2.4.2. Convolutional Neural Network

Convolutional neural network (CNN) merupakan suatu arsitektur DL yang terinspirasi oleh mekanisme persepsi visual makhluk hidup. Pada 1959, Hubel dan Wiesel [27] menemukan sel tersebut pada korteks visual hewan yang bertanggung jawab untuk mendeteksi cahaya di bidang reseptif. Terinspirasi oleh hal ini,

Kunihiko Fukushima mengusulkan neocognitron pada 1980 [28], yang dapat dianggap sebagai pendahulu CNN. Pada 1990, LeCun dkk. [29] menerbitkan makalah untuk membangun kerangka kerja modern CNN, dan kemudian memperbaruinya dalam jurnalnya yang berjudul "Gradient-based learning applied to document recognition". Mereka mengembangkan multi-layer artificial neural network yang disebut LeNet-5 yang dapat mengklasifikasikan digit tulisan tangan. Seperti neural networks yang lainnya, LeNet-5 memiliki banyak lapisan dan dapat dilatih dengan algoritma *backpropagation* [26]. Hal itu dapat memperoleh representasi yang efektif dari gambar asli, yang membuatnya mungkin untuk mengenali pola visual secara langsung dari piksel mentah dengan sedikit pra-pemrosesan.

2.4.2.1. Arsitektur LeNet-5

Ada banyak varian arsitektur CNN dalam literatur. Namun, komponen dasarnya sangat mirip. Jika kita melihat contoh arsitektur LeNet-5 yang terkenal, arsitektur mereka terdiri dari tiga jenis lapisan, yaitu *convolutional*, *pooling*, dan lapisan yang terhubung sepenuhnya. Lapisan *convolutional* bertujuan untuk mempelajari representasi fitur dari input. Untuk penjelasan lebih lanjut, *convolution* dijelaskan pada BAB II bagian 2.2.2. Gambar 2.8 merupakan arsitektur LeNet-5, dimana lapisan konvolusi terdiri dari beberapa kernel konvolusi yang digunakan untuk menghitung *feature maps* yang berbeda.



Gambar 2. 8 Arsitektur LeNet-5

Secara khusus, setiap neuron dari *feature maps* terhubung dengan wilayah ketetanggaan neuron di lapisan sebelumnya. Ketetanggaan seperti itu disebut

sebagai neuron bidang reseptif pada lapisan sebelumnya. *Feature maps* yang baru dapat diperoleh terlebih dahulu dengan menggabungkan input dengan kernel yang dipelajari dan kemudian menerapkan fungsi aktivasi *non-linear* berdasarkan elemen pada hasil yang digabungkan. Harus diperhatikan bahwa untuk menghasilkan setiap *feature maps*, kernel digunakan bersama oleh semua lokasi *input spasial*. *Feature maps* yang lengkap diperoleh dengan menggunakan beberapa kernel yang berbeda. Secara matematis, nilai fitur pada lokasi (i, j) di *feature maps k* dari lapisan l, $\boldsymbol{Z}_{i,j,k}^{l}$ memiliki persamaan:

$$Z_{i,j,k}^{l} = {W_k^l}^T x_{i,j}^{l} + b_k^l$$

(2. 14)

Di mana:

l = Lapisan ke-*l*.

k = Filter.

W_k^l = Bobot matriks.

b_k^l = *Bias* dari filter *k* dari lapisan *l*.

$x_{i,j}^{l}$ = *Input patch* yang berpusat di lokasi (i, j) dari lapisan l.

Perlu diingat bahwa kernel w_k^l yang menghasilkan *feature maps* $z_{:,:,k}^{l}$ dibagikan. Mekanisme pembagian berat seperti itu memiliki beberapa keunggulan seperti bisa mengurangi kompleksitas model dan membuat *network* lebih mudah untuk dilatih. Fungsi aktivasi memperkenalkan *nonlinear* ke CNN, yang diinginkan untuk *multi-layer networks* untuk mendeteksi fitur *nonlinear*. Anggaplah $a(z_{i,j,k}^{l})$ menunjukkan fungsi aktivasi *nonlinear*. Nilai aktivasi $a_{i,j,k}^{l}$ dari fitur konvolusional $z_{i,j,k}^{l}$ bisa dihitung sebagai:

$$a_{i,j,k}^{l} = a(z_{i,j,k}^{l})$$

(2. 15)

Di mana:

a = Fungsi aktivasi.

Fungsi aktivasi yang biasa digunakan adalah *Sigmoid*, *tanh*, ReLU, dan *softmax*. Lapisan *pooling* bertujuan untuk mencapai *shift-invariance* dengan mengurangi resolusi *feature maps*. Hal ini biasanya ditempatkan di antara dua lapisan konvolusi. Setiap *feature maps* dari lapisan *pooling* terhubung ke *feature maps* sebelumnya yang sesuai lapisan konvolusi. Untuk setiap *feature maps* $a^{l}_{:,:,k}$ kita memiliki:

$$y^{l}_{i,j,k} = pool\left(a^{l}_{m,n,k}\right), \forall (m,n) \epsilon R_{ij}$$

(2. 16)

Di mana:

$a^{l}_{m,n,k}$ = *Tensor* k dimensi dengan ukuran $m \times n$.

R_{ij} = Himpunan atau rentang nilai yang mungkin untuk indeks (m, n)

Secara detail, *pooling* adalah operasi di bidang CNN, berperan penting dalam mencari invariant translasi dan mengurangi dimensi, sehingga mengurangi tuntutan komputasi dan mengurangi *overfitting*. *Pooling* melakukan operasi pengambilan sampel turun di sepanjang dimensi spasial (lebar dan tinggi), menghasilkan *feature maps* terkondensasi yang masih menyimpan informasi yang paling menonjol. Pengurangan ini tidak hanya mengurangi biaya komputasi untuk lapisan berikutnya, tetapi juga membantu mencegah model agar tidak terlalu dekat dengan data pelatihan, yang dapat secara tidak sengaja menangkap *noise* dan *outlier*.

Dua metode *pooling* yang umum termasuk *max pooling* dan *average pooling*. Masing-masing prosedur ini biasanya diterapkan pada wilayah input *feature maps*

(disebut dengan “bidang reseptif”), yang mungkin tumpang tindih dengan wilayah lain tergantung pada set panjang langkah. *Max pooling* dapat ditulis sebagai berikut:

$$y_{i,j,k}^{l} = \underset{m,n \in R_{i,j}}{\text{argmax}}(a_{m,n,k}^{l})$$

(2. 17)

Di mana:

$a_{m,n,k}^{l}$ = *Tensor* k dimensi dengan ukuran $m \times n$.

$\underset{m,n \in R_{i,j}}{\text{argmax}}$ = Nilai maksimum pada kolom (m, n) di mana (m, n) merupakan bilangan *real* pada kolom (i, j)

Max pooling, seperti namanya, mengekstrak nilai maksimum dari setiap bidang reseptif, membuang semua nilai lainnya. Metode ini didasarkan pada asumsi bahwa nilai maksimum adalah representasi fitur lokal yang paling relevan di bidang reseptif. Manfaat *max pooling* terletak pada kemampuannya untuk mempertahankan fitur yang paling signifikan sekaligus mengurangi dimensi, sehingga memastikan retensi informasi pola kritis. Namun, itu tidak mempertahankan representasi informasi holistik di bidang reseptif asli, yang dapat menyebabkan hilangnya informasi.

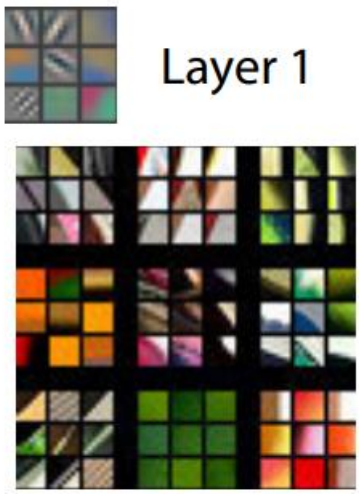
2.4.2.2. Visualisasi dan Memahami Jaringan Konvolusional

Arsitektur DL, khususnya CNN, telah dipuji karena kinerjanya yang luar biasa dalam berbagai tugas mulai dari klasifikasi gambar hingga pemrosesan bahasa alami. Komponen penting dari kemanjurannya adalah kapasitas pembelajaran fitur hierarkis, yang secara berurutan terbentang di berbagai lapisan. Lapisan bukan hanya unit pemrosesan, tetapi lapisan abtraksi yang tajam, dengan setiap lapisan ditugaskan untuk mempelajari representasi fitur tertentu.

Pada lapisan awal, sering disebut sebagai lapisan ‘tingkat rendah’, CNN sebagian besar belajar untuk mendeteksi fitur sederhana dan dasar seperti tepi,

sudut, dan kontras warna. *Filter* yang diterapkan di sini pada dasarnya membantu mengidentifikasi susunan geometris utama dan gumpalan warna pada gambar. Prinsip dasar yang bekerja adalah bidang reseptif lokal, memungkinkan jaringan untuk fokus pada area lokal dari gambar masukan. Akibatnya, deteksi pola pada tahap ini cukup primitif namun sangat diperlukan untuk pengenalan fitur tingkat tinggi.

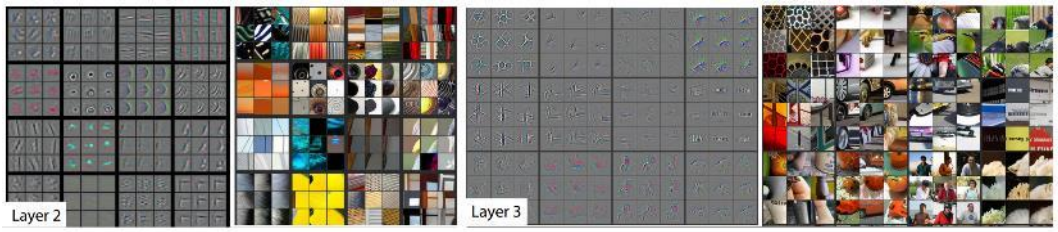
Gambar yang disajikan dalam jurnal yang dibuat oleh Zeiler dan Fergus (2014), berjudul "*Visualizing and Understanding Convolutional Networks*" [30], memfasilitasi pemahaman mendalam tentang proses pembelajaran hierarkis di CNN. Masing-masing gambar ini menawarkan visualisasi mendetail dari fitur spesifik yang ditangkap oleh setiap lapisan dalam CNN, sehingga memberikan wawasan kritis ke dalam mekanisme DL. Gambar 2.9 mengilustrasikan kapasitas pembelajaran lapisan pertama di CNN.



Gambar 2. 9 Ilustrasi Hal yang Dipelajari CNN Lapisan Pertama

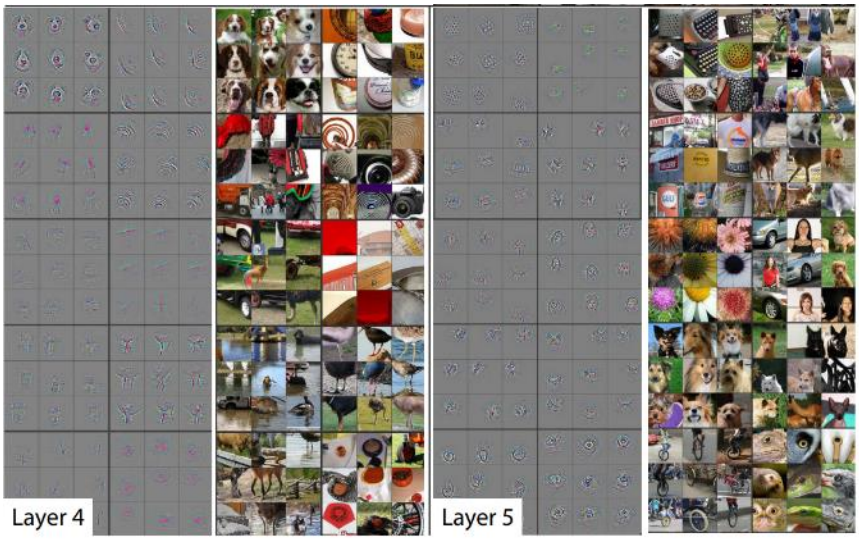
Seperti yang digambarkan, lapisan ini membedakan fitur dasar seperti tepi, kontras warna, dan tekstur dasar. Neuron individu di lapisan ini pada dasarnya berfungsi sebagai detektor tepi yang berorientasi. Setiap subgambar pada gambar 2.9 sesuai dengan sembilan aktivasi *filter* teratas, menyoroti kekhususan deteksi tepi dan tekstur. Pengkodean warna di setiap subgambar mewakili nilai absolut

bobot, di mana warna yang lebih hangat menandakan nilai yang lebih tinggi. Ini menyiratkan bahwa neuron pada lapisan ini bereaksi terhadap kontras warna dan pola tekstur sederhana pada gambar masukan. Gambar 2.10 merangkum proses pembelajaran dari lapisan kedua dan ketiga dari CNN.



Gambar 2. 10 Ilustrasi Proses Pembelajaran Lapisan Kedua dan Ketiga CNN

Berdasarkan gambar 2.10, Lapisan kedua, seperti yang dapat diamati pada gambar, menggabungkan fitur tingkat rendah yang diidentifikasikan pada lapisan pertama untuk membedakan pola dan bentuk yang lebih kompleks. Menarik juga untuk dicatat munculnya tanda-tanda pertama skala dan invariant pada tahap ini. Lapisan ketiga CNN, seperti yang digambarkan pada bagian kedua gambar, selanjutnya menggabungkan struktur bagian ini untuk membentuk pola dan struktur yang lebih kompleks, yang mulai menyerupai bagian objek yang dapat diidentifikasi. Lapisan ketiga CNN, seperti yang digambarkan pada bagian kedua gambar, selanjutnya menggabungkan struktur bagian ini untuk membentuk pola dan struktur yang lebih kompleks, yang mulai menyerupai bagian objek yang dapat diidentifikasi. Terakhir, gambar 2.11 menjelaskan proses pembelajaran pada lapisan keempat dan kelima.



Gambar 2. 11 Ilustrasi Hal yang Dipelajari CNN Lapisan Keempat dan Kelima

Pada tahap gambar 2.11, lapisan keempat menggabungkan fitur tingkat menengah untuk memahami objek dan pemandangan yang lebih kompleks, dengan beberapa *neuron* berspesialisasi dalam konsep yang lebih rumit seperti kisi dan pola berulang. Lapisan kelima mewakili puncak dari proses pembelajaran hierarkis ini. Ini mengasimilasi semua fitur tingkat rendah untuk memahami dan mempelajari struktur objek lengkap, sehingga mampu mendeteksi konsep tingkat tinggi seperti wajah atau teks dalam berbagai perspektif dan kondisi pencahayaan.

Intinya, angka-angka ini memberikan panduan ilustratif untuk pembelajaran hierarkis yang dilakukan oleh CNN, yang secara efektif memetakan perjalanan dari deteksi tepi yang belum sempurna ke pengenalan objek yang komprehensif. Kerangka pembelajaran hierarkis ini mendasari kinerja CNN yang luar biasa dalam berbagai aplikasi, mulai dari pengenalan gambar hingga pemrosesan bahasa alami.

2.4.2.3. Batch Normalization

Batch normalization adalah metode yang dirancang untuk mengatasi permasalahan "*internal covariate shift*" dalam proses pelatihan model jaringan saraf dalam, yang dikemukakan oleh Ioffe Szegedy [31]. Pergeseran dalam

distribusi antar lapisan bisa mengganggu langkah-langkah belajar sebelumnya dan memperlambat konvergensi proses pelatihan.

Metode ini bekerja dengan cara normalisasi *output* lapisan sebelumnya, di mana rata-rata dan varians dari *output* tersebut dihitung dan digunakan untuk menskalakan dan menggeser distribusi *output* tersebut sehingga memiliki rata-rata nol dan varians satu [31]. Operasi ini diterapkan untuk setiap *mini-batch*, yaitu sekelompok sampel yang dipilih secara acak dari dataset utuh dalam setiap iterasi pelatihan.

Efektivitas *batch normalization* dalam meningkatkan efisiensi pelatihan model *deep learning*, mereduksi kebutuhan terhadap regulasi yang ketat, dan mempercepat konvergensi telah ditunjukkan dalam penelitian [31]. Selain itu, metode ini juga memungkinkan penggunaan fungsi aktivasi yang sebelumnya sulit untuk dioptimalkan.

Batch normalization juga berperan sebagai regularisasi, meski bukan tujuan awal dari metode ini, tetapi efek ini terbukti dalam penelitian empiris [31].

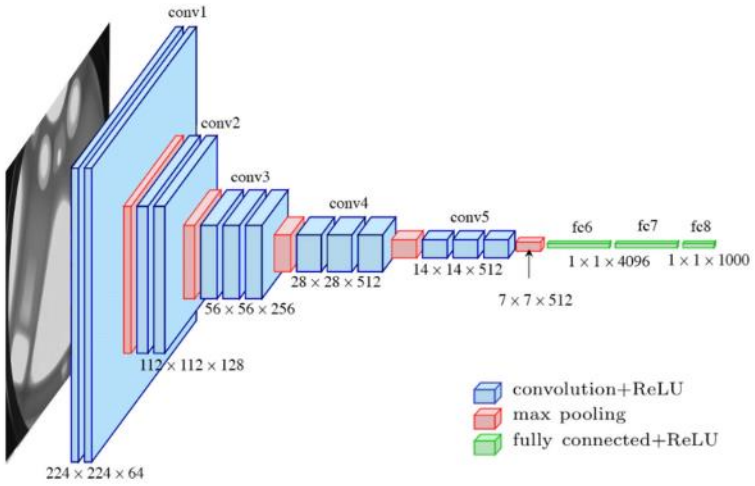
2.4.2.4. VGG

VGG, atau *Visual Geometry Group*, adalah nama dari kelompok penelitian yang mengembangkan model arsitektur *deep learning* yang dikenal sebagai VGGNet. Model ini pertama kali diperkenalkan oleh K. Simonyan dan A. Zisserman, dalam sebuah jurnal yang berjudul "*Very Deep Convolutional Networks for Large-Scale Image Recognition*" [32].

VGGNet adalah sebuah arsitektur jaringan saraf konvolusional yang dirancang untuk pengenalan citra pada skala besar. Keunikan utama dari VGGNet adalah kedalaman arsitekturnya, dimana terdapat 16 hingga 19 lapisan yang terdiri dari lapisan konvolusi dan *fully connected*.

VGGNet memanfaatkan filter konvolusi berukuran 3×3, yang merupakan ukuran filter yang paling kecil yang masih mempertahankan informasi spasial. Penggunaan filter ini memungkinkan VGGNet untuk memiliki banyak lapisan

dengan jumlah parameter yang relatif lebih sedikit dibandingkan dengan model yang menggunakan filter konvolusi berukuran lebih besar. VGGNet juga menggunakan fungsi aktivasi ReLU setelah setiap lapisan konvolusi untuk mempercepat proses pelatihan dan membantu model dalam mempelajari fitur *non-linear*. Gambar 2.12 merupakan ilustrasi dari arsitektur VGG16.



Gambar 2. 12 Ilustrasi Arsitektur VGG16

Gambar 2.12 menunjukkan arsitektur dari model VGG16, sebuah model jaringan saraf konvolusional yang mendalam. Arsitektur VGG16 terdiri dari 13 lapisan konvolusional, yang diikuti oleh 3 lapisan sepenuhnya (*fully-connected*). Lapisan konvolusional dibagi menjadi 5 blok, di mana setiap blok diakhiri dengan lapisan *pooling* maksimum untuk mengurangi dimensi spasial. Jumlah filter dalam lapisan konvolusional meningkat dengan peningkatan kedalaman jaringan, dimulari dari 64 dan berakhir hingga 512.

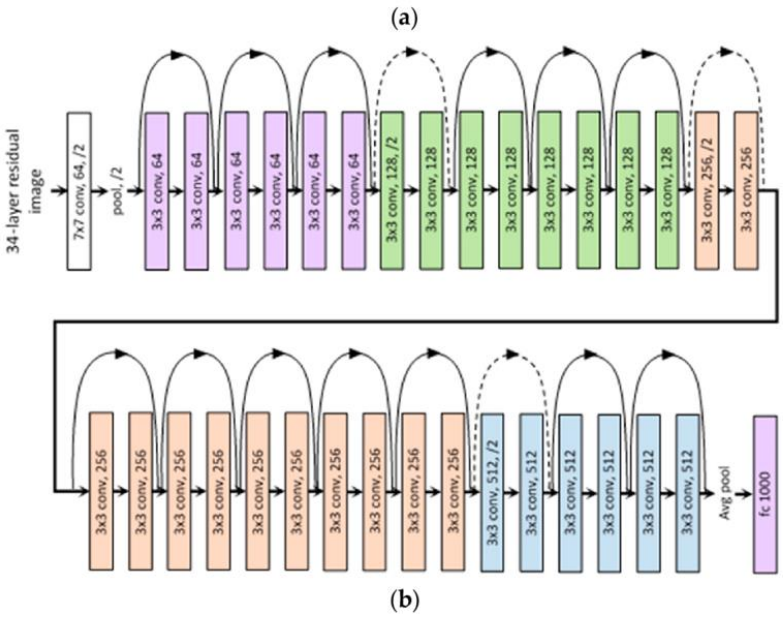
Lapisan sepenuhnya terhubung di akhir jaringan terdiri dari dua lapisan dengan 4096 neuron dan lapisan output dengan jumlah neuron yang sama dengan jumlah kleas dalam tugas klasifikasi. Fungsi aktivasi ReLU digunakan di semua lapisan kecuali lapisan *output*, di mana *softmax* digunakan untuk menghasilkan probabilitas kelas.

Meski model ini memerlukan banyak sumber daya komputasi dan memori yang besar, keunggulan utama dari VGGNet adalah fakta bahwa model ini memiliki struktur yang sangat seragam, yang membuatnya mudah untuk diimplementasikan dan dimodifikasi. Oleh karena itu, VGGNet menjadi acuan penting dalam pengembangan arsitektur *deep learning* yang lebih baru.

2.4.2.5. Residual Network

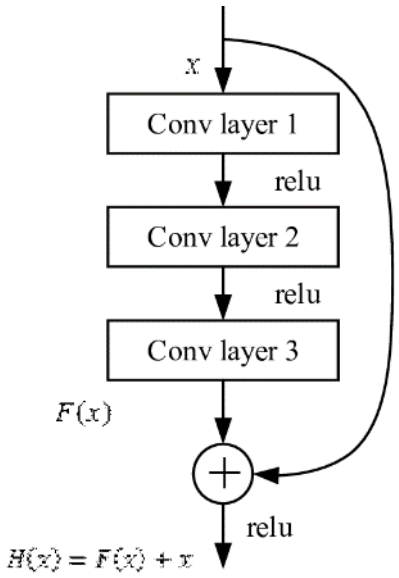
Residual Network atau ResNet merupakan suatu arsitektur jaringan saraf dalam yang dikemukakan pada jurnal [33]. ResNet dirancang untuk menangani masalah yang biasa terjadi dalam pelatihan jaringan saraf dalam, yaitu *vanishing gradient* dan *exploding gradient*. Permasalahan ini muncul ketika jaringan menjadi sangat dalam dan menyebabkan gradien melalui lapisan tersebut menjadi sangat kecil atau sangat besar, sehingga mempengaruhi proses pembelajaran.

ResNet memperkenalkan konsep yang dikenal dengan *skip connection* atau *shortcut connection* untuk mengatasi permasalahan tersebut. Pada dasarnya, *skip connection* memungkinkan sinyal dari lapisan sebelumnya untuk melompati satu atau lebih lapisan dan langsung masuk ke lapisan selanjutnya. Dengan kata lain, *output* dari lapisan sebelumnya ditambahkan ke *output* dari lapisan yang dilompati. Ini memungkinkan gradien untuk mengalir langsung melalui *skip connection* tanpa melewati lapisan mana pun, yang membantu memitigasi masalah *vanishing gradient*. Gambar 2.13 merupakan ilustrasi arsitektur *Residual Network*.



Gambar 2. 13 Arsitektur Residual Network

Gambar 2.13 menampilkan arsitektur dari ResNet. Arsitektur ini terdiri dari beberapa blok residual, di mana setiap blok residual memiliki setidaknya dua lapisan konvolusional dan sebuah *skip connection* yang menghubungkan input blok ke *outputnya*. Fungsi aktivasi ReLU digunakan setelah setiap lapisan konvolusional, dan normalisasi *batch* digunakan sebelum aktivasi. Gambar 2.14 merupakan ilustrasi dari *skip connection*.



Gambar 2. 14 Ilustrasi Skip Connection

Gambar 2.14 memberikan ilustrasi lebih detail tentang *skip connection*. Dalam blok residual, input x ditambahkan ke *output* dari rangkaian lapisan konvolusional sebelum diteruskan ke fungsi aktivasi. Hal ini dapat dirumuskan sebagai:

$$H(x) = F(x) + x \tag{2. 18}$$

Di mana:

$F(x)$ = Lapisan konvolusional.

$H(x)$ = *Output* blok residual.

Skip connection ini memungkinkan model untuk belajar fungsi identitas, yang membantu dalam mengatasi masalah *vanishing gradient* saat melatih jaringan yang sangat dalam.

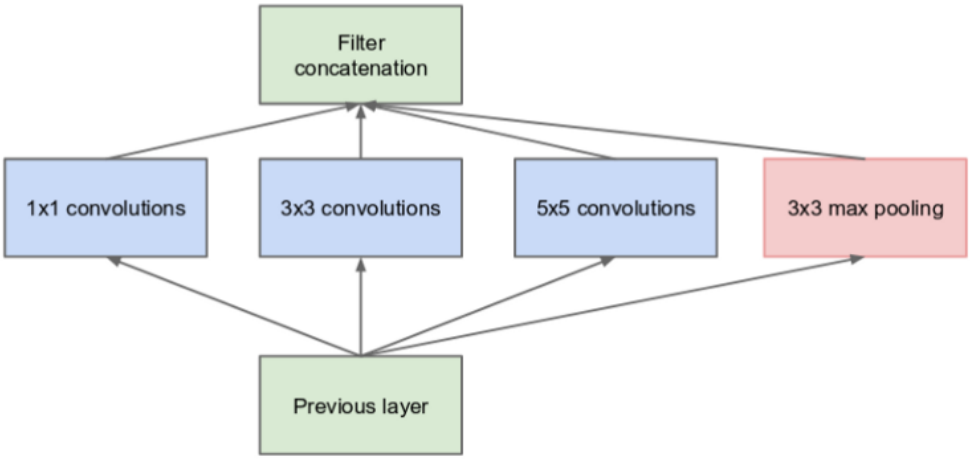
Arsitektur ResNet telah menunjukkan kinerja yang sangat baik dalam berbagai tantangan pengenalan gambar, dan memenangkan kompetisi ILSVRC (*ImageNet Large Scale Visual Recognition Challenge*) pada tahun 2015. Penggunaan *skip connection* telah menjadi konsep yang sangat berpengaruh dalam pengembangan arsitektur jaringan saraf dalam selanjutnya.

2.4.2.6. Inception Network

Inception network adalah salah satu jenis arsitektur jaringan saraf tiruan yang mendalam yang dirancang untuk melakukan pengenalan gambar. Arsitektur ini pertama kali diperkenalkan oleh Christian Szegedy dan rekan-rekannya dari Google dalam jurnal mereka yang berjudul "*Going Deeper with Convolutions*" pada tahun 2014 [34].

Inception network dirancang untuk mengatasi masalah efisiensi komputasi dalam jaringan saraf tiruan yang mendalam. Dalam arsitektur ini, operasi konvolusi dilakukan pada berbagai skala secara paralel sebelum hasilnya digabungkan. Ini memungkinkan jaringan untuk belajar fitur pada berbagai tingkat abstraksi dari data, yang dapat meningkatkan kinerja pengenalan gambar.

Struktur dasar dari *inception network* adalah modul *inception*, yang terdiri dari empat cabang paralel. Gambar 2.15 merupakan ilustrasi dari model *inception*.



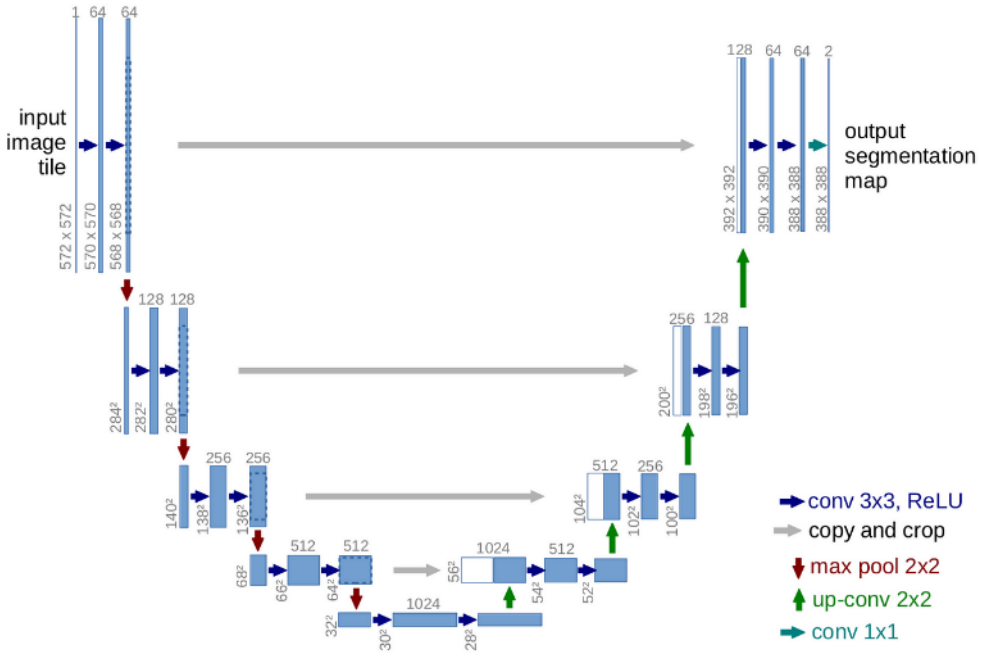
Gambar 2. 15 Modul Inception

Seperti yang dapat dilihat dari gambar 2.15, setiap cabang melakukan operasi konvolusi dengan ukuran *kernel* yang berbeda (1×1, 3×3, dan 5×5) atau operasi *pooling*. Hasil dari setiap cabang kemudian digabungkan menjadi satu volume *output*.

2.4.2.7. U-Net

U-Net adalah salah satu arsitektur jaringan saraf dalam CNN yang dirancang khusus untuk segmentasi citra medis, yang pertama kali diperkenalkan oleh Ronneberger et al. [35]. Arsitektur U-Net mendapatkan namanya dari bentuk “U” yang dihasilkan oleh dua bagian utama arsitektur: bagian penyandian (*encoding*)

dan bagian penyandian balik (*decoding*). Gambar 2.16 merupakan ilustrasi dari arsitektur U-Net.



Gambar 2. 16 Arsitektur U-Net

Gambar 2.16 menunjukkan bagian penyandian terdiri dari serangkaian konvolusi dan *pooling* maksimum yang berfungsi untuk mengekstrak dan menurunkan dimensi fitur dalam citra. Setelah mencapai titik terendah dalam bentuk "U", proses kemudian dibalik dalam bagian penyandian balik, di mana serangkaian operasi *up-sampling* dan konvolusi dilakukan untuk memperoleh Kembali resolusi spasial yang asli.

Salah satu aspek penting dalam arsitektur U-Net adalah penggunaan *skip connections* antara lapisan penyandian balik yang setara. *Skip connections* ini menggabungkan fitur tingkat tinggi yang diekstrak dalam bagian penyandian dengan fitur yang disandikan balik, yang memungkinkan jaringan untuk menggunakan informasi kontekstual dan lokal dalam membuat prediksi segmentasi.

Arsitektur U-Net telah terbukti efektif dalam tugas segmentasi citra medis, di mana objek yang harus disegmentasi dapat sangat bervariasi dalam bentuk dan ukuran. Hal ini membuat arsitektur ini menjadi pilihan populer dalam banyak aplikasi di bidang pengolahan citra.

Upsampling, dalam konteks U-Net, terjadi dalam bagian penyandian balik. Di sini, fitur dari bagian penyandian yang telah mengalami *downsampling* dibalik menjadi resolusi spasial yang lebih tinggi. Prosedur ini sering kali dilakukan menggunakan operasi *up-convolution* (atau *transposed convolution*) atau operasi *up-sampling* yang lebih sederhana diikuti oleh konvolusi.

Fitur-fitur ini kemudian digabungkan dengan fitur yang sesuai dari bagian penyandian melalui mekanisme *skip connection*, memungkinkan model untuk memanfaatkan informasi kontekstual dan spasial dalam skala yang berbeda-beda. Ini penting dalam tugas-tugas seperti *super-resolution*, di mana detail tingkat tinggi dari gambar asli perlu dipertahankan sambil meningkatkan resolusi.

Aplikasi U-Net dalam konteks *super-resolution* dapat ditemukan dalam berbagai penelitian, seperti yang ditunjukkan pada jurnal [36]. Di sini, mereka mengembangkan metode berbasis U-Net yang mereka namakan SRU-Net mampu menghasilkan hasil yang lebih baik dibandingkan dengan metode sebelumnya dalam hal akurasi dan efisiensi komputasional.

2.4.3. Permasalahan Pada *Deep Learning*

Terdapat beberapa permasalahan yang sering dihadapi ketika mengimplementasi *deep learning*. Di bawah merupakan salah satu permasalahan umum yang sering dihadapi para peneliti:

2.4.3.1. Overfitting dan Underfitting

Overfitting dan *underfitting* adalah dua fenomena yang dapat terjadi saat melatih model *machine learning* atau jaringan saraf tiruan. *Overfitting* terjadi ketika model menjadi terlalu kompleks atau ketika model dilatih terlalu lama, sehingga model mulai mempelajari detail dan *noise* dalam data pelatihan hingga tingkat yang

tidak diinginkan. Akibatnya, model memiliki performa yang sangat baik pada data pelatihan tetapi performa yang buruk pada data yang belum pernah dilihat sebelumnya (data validasi atau pengujian). Dengan kata lain, model tidak mampu menggeneralisasi dengan baik dari data pelatihan ke data baru [37].

Sebaliknya, *underfitting* terjadi ketika model terlalu sederhana untuk menangkap struktur dan pola yang ada dalam data. Model *underfitting* mungkin tidak memiliki kapasitas yang cukup untuk mempelajari data, atau mungkin tidak dilatih cukup lama untuk mempelajari pola yang ada. Akibatnya, model memiliki performa yang buruk baik pada data pelatihan maupun data validasi atau pengujian [38].

2.4.3.2. Vanishing Gradient

Masalah *vanishing gradient* adalah fenomena yang sering terjadi dalam pelatihan jaringan saraf tiruan yang dalam, di mana gradien yang digunakan dalam pembaruan bobot selama *backpropagation* menjadi sangat kecil, hampir nol. Hal ini mengakibatkan bobot dalam lapisan awal jaringan (yaitu, lapisan yang lebih dekat ke input) hampir tidak diperbarui, yang pada gilirannya menghambat pelatihan dan konvergensi model [39].

Untuk mendeteksi masalah ini, kita dapat memantau gradien selama proses pelatihan. Jika gradien menjadi sangat kecil dan bobot dalam lapisan awal hampir tidak berubah, maka kita mungkin menghadapi masalah *vanishing gradient*.

Ada beberapa pendekatan untuk mengatasi masalah *vanishing gradient*. Salah satunya adalah penggunaan fungsi aktivasi yang dirancang untuk mengurangi masalah ini, seperti ReLU [40]. Selain itu, teknik normalisasi seperti *batch normalization* juga dapat membantu [41]. Arsitektur jaringan juga dapat membantu, salah satunya adalah *residual networks* (ResNets) yang memperkenalkan koneksi *skip* atau *shortcut* antara lapisan [42].

2.4.3.3. Exploding Gradient

Masalah *exploding gradient* adalah fenomena yang dapat terjadi dalam pelatihan jaringan saraf tiruan yang dalam, di mana gradien yang digunakan dalam pembaruan bobot selama *backpropagation* menjadi sangat besar. Hal ini dapat mengakibatkan bobot dalam jaringan menjadi sangat besar atau sangat kecil, yang pada gilirannya dapat menyebabkan model menjadi tidak stabil dan sulit untuk dilatih [43].

Seperti permasalahan *vanishing gradient*, untuk mendeteksi masalah ini, kita dapat memantau gradien selama proses pelatihan. Jika gradien sangat besar dan bobot dalam jaringan berubah secara drastic, maka kita mungkin menghadapi masalah *exploding gradient*.

Ada beberapa pendekatan untuk mengatasi masalah *exploding gradient*. Salah satunya adalah penggunaan teknik yang dikenal sebagai *gradient clipapping*, yang membatasi nilai gradien agar tidak melebihi ambang batas tertentu [44]. Selain itu, penggunaan fungsi aktivasi yang tepat dan teknik normalisasi juga dapat membantu [40][41].

2.5. *Self-Supervised Learning*

Self-supervised learning adalah paradigma *machine learning* yang memanfaatkan data yang tidak berlabel untuk belajar representasi yang berguna dari data tersebut. Dalam konteks ini, model belajar untuk memprediksi bagian dari data berdasarkan bagian lainnya, dengan ide bahwa proses ini memaksa model untuk mempelajari struktur pola yang ada dalam data. Dalam banyak kasus, *self-supervised learning* telah menunjukkan hasil yang menjanjikan dalam belajar representasi yang berguna dari data yang besar dan tidak berlabel [45][46].

Ide awal dari *self-supervised learning* muncul dari kebutuhan untuk mengurangi ketergantungan pada data berlabel yang mahal dan sulit untuk diperoleh dalam jumlah besar. Dengan memanfaatkan data yang tidak berlabel, yang biasanya lebih mudah dan murah untuk diperoleh, *self-supervised learning*

menawarkan alternatif yang menarik untuk pendekatan pembelajaran yang diawasi secara tradisional [47].

Ada beberapa teknik yang digunakan dalam *self-supervised learning*, termasuk *generative models*, *contrastive learning*, dan *mutual information maximization*. Teknik-teknik ini dirancang untuk memaksimalkan informasi yang dapat diperoleh dari data yang tidak berlabel dan untuk belajar representasi yang berguna dari data tersebut [48].

2.6. GAN

Deep learning merupakan domain yang mengandung potensi luar biasa dan mampu menyelesaikan isu-isu yang sebelumnya dianggap tidak teratasi. Namun, dalam beberapa tahun terakhir, kemampuan *deep learning* secara utama terbatas pada prediksi kelas dalam data berdimensi tinggi. Aspek yang paling menonjol dari *deep learning* sebelum kemunculan *Generative Adversarial Networks* (GANs) adalah prediksi label pada data mentah yang tidak memiliki kelas yang ditentukan sebelumnya. Hingga saat ini, prestasi yang paling signifikan dalam domain *deep learning* adalah penggunaan algoritma yang melibatkan model diskriminatif, yang menerima input berdimensi tinggi dan menghasilkan prediksi [49][50].

Tujuan utama dari GAN adalah untuk mengembangkan model yang tidak hanya mampu mengklasifikasikan data input, tetapi juga mampu menghasilkan *output* berupa gambar. GAN telah digunakan dalam berbagai aplikasi inovatif, termasuk pembuatan gambar sintetis dari individu yang tidak ada dalam realita, konversi gambar *grayscale* menjadi berwarna, peningkatan resolusi gambar, dan lainnya. Mekanisme kerja GAN dapat diilustrasikan melalui gambar 2.17 berikut.

Gambar 2. 17 Ilustrasi Cara Kerja GAN

Dalam konteks formal, GAN merupakan model jaringan adversarial, di mana model generatif berkompetisi dengan model adversarial [51]. Model generatif berusaha untuk menentukan apakah sampel tertentu berasal dari distribusi model atau distribusi data. Untuk mempermudah pemahaman, gambar 2.17 menyajikan analogi dari GAN. Model generatif dapat dianggap sebagai penipu yang mencoba membuat lukisan palsu, sedangkan model adversarial berperan sebagai detektif yang berusaha membedakan antara lukisan asli dan palsu. Selama fase pelatihan, model generatif mencoba membuat lukisan, tetapi pada awalnya hanya mampu menghasilkan *noise* acak karena bobot yang belum optimal. Model adversarial kemudian membandingkan gambar yang dihasilkan oleh model generatif dengan gambar asli, mencoba membedakannya, dan berusaha menebak mana yang asli. Pada awalnya, model adversarial dapat dengan mudah menebak gambar asli, tetapi seiring berjalannya pelatihan, model generatif menjadi semakin baik, sehingga model adversarial mengalami kesulitan dalam menebak gambar asli.

Alasan utama keberhasilan GAN dalam menghasilkan gambar sintetis adalah karena model generatif pada akhirnya mampu mengembangkan gambar palsu yang sangat mirip dengan gambar asli, sehingga model adversarial terpaksa membuat prediksi secara acak dengan probabilitas $^1/_2$. Penting untuk dicatat bahwa kedua model, generatif dan adversarial, dimulai dari titik awal di mana keduanya belum

terlatih dan membuat prediksi secara acak, hingga kedua model tersebut dilatih dan ditingkatkan.

Fungsi kerugian dari model adversarial dapat didefinisikan sebagai berikut:

$$Adversarial = \frac{1}{m}\sum_{i=1}^{m}\left[log\, D\left(X^{(i)}\right) + log\left(1 - D\left(G\left(z^{(i)}\right)\right)\right)\right]$$

(2. 19)

Di mana:

$D\left(X^{(i)}\right)$ = *Output* dari model adversarial (D) ketika diberikan sampel data asli $X^{(i)}$.

$D\left(G\left(z^{(i)}\right)\right)$ = *Output* dari model adversarial ketika diberikan sampel data palsu yang dihasilkan oleh model generatif $D\left(G\left(z^{(i)}\right)\right)$.

Sedangkan, untuk fungsi kerugian model generatif dapat didefinisikan sebagai berikut:

$$Generatif = \frac{1}{m}\sum_{i=1}^{m} log\left(1 - D\left(G\left(z^{(i)}\right)\right)\right)$$

(2. 20)

Di mana:

$D\left(G\left(z^{(i)}\right)\right)$ = Hal serupa seperti pada rumus 2.19.

Dapat dilihat bahwa fungsi kerugian dari model adversarial mirip dengan fungsi kerugian pada klasifikasi biner. $X^{(i)}$ adalah bilangan biner (0 atau 1), dan model adversarial berusaha untuk meminimalkan $X^{(i)}$. Dalam persamaan $log\left(1 - D\left(G\left(z^{(i)}\right)\right)\right)$, $G\left(z^{(i)}\right)$ adalah fungsi dari model generatif dan $z^{(i)}$ adalah

noise acak. Dari perspektif model adversarial, kita ingin $D\left(G\left(z^{(i)}\right)\right)$ bernilai 0, yang berarti adversarial berusaha memaksimalkan fungsi kerugian generatif.

Di sisi lain, model generatif berusaha menipu model adversarial dengan meminimalkan fungsi kerugian generatif. Model generatif menginginkan nilai $D\left(G\left(z^{(i)}\right)\right)$ mendekati 1, yang akan memaksimalkan fungsi kerugian dari model adversarial.

Berdasarkan penjelasan di atas, kita dapat menggabungkan kedua fungsi kerugian dari model di atas menjadi satu ekspresi [51]:

$$\min_G \max_D V(D,G) = \mathbb{E}_{z\sim Pdata(x)}[\log D(x)] + \mathbb{E}_{z\sim Pz(z)}\left[\log\left(1 - D\left(G\left(z^{(i)}\right)\right)\right)\right] \quad \textit{(2. 21)}$$

Di mana:

$\mathbb{E}_{z\sim Pdata(x)}[\log D(x)]$	= nilai ekspektasi dari logaritma *output* model adversarial ketika diberikan sampel data asli.
$\mathbb{E}_{z\sim Pz(z)}\left[\log\left(1 - D\left(G\left(z^{(i)}\right)\right)\right)\right]$	= Nilai ekspektasi dari logaritma satu dikurangi *output* model adversarial ketika diberikan sampel data palsu.

Selama beberapa tahun ini, GAN telah menjadi pusat perhatian dalam dunia penelitian kecerdasan buatan sejak diperkenalkan. Meskipun telah menunjukkan kapabilitas yang menjanjikan, banyak aspek dari GAN yang masih memerlukan eksplorasi dan perbaikan lebih lanjut. Hal ini menjadikan GAN sebagai bidang yang kontinu dan intensif didekati oleh para peneliti.

2.6.1. SRGAN

Super-Resolution Generative Adversarial Network (SRGAN) [8] merupakan sebuah pendekatan inovatif dalam bidang pemulihan citra beresolusi tinggi dari citra beresolusi rendah. SRGAN memanfaatkan jaringan adversarial untuk

menghasilkan citra yang fotorealistik dengan faktor peningkatan hingga $4\times$. SRGAN merupakan salah satu teknik terdahulu yang menggunakan gambar *ground-truth* yang disebut dengan gambar *high resolution* (HR). Lalu, gambar tersebut akan dilakukan data augmentasi yang akan menghasilkan gambar *low resolution* (LR). SRGAN merupakan teknik krusial untuk melakukan *image enhancement*, dimana kita ingin meningkatkan kualitas piksel pada gambar.

2.6.2. StyleGAN

Style-Based Generator Adversarial Network (StyleGAN) [52] mewakili perkembangan penting dalam bidang generasi citra. Model ini memanfaatkan pendekatan berbasis gaya untuk menghasilkan citra yang memiliki variasi dan detail yang kaya. Salah satu keunggulan utama dari StyleGAN adalah kemampuannya untuk memodelkan dan menghasilkan citra dengan tekstur dan karakteristik yang kompleks, yang sering kali sulit dicapai dengan model generatif lainnya. Dalam konteks generasi citra, StyleGAN memanfaatkan vektor gaya yang memungkinkan kontrol yang lebih besar atas aspek-aspek tertentu dari citra yang dihasilkan, seperti tekstur, warna, dan bentuk. Hal ini memungkinkan peneliti dan praktisi untuk menghasilkan citra yang tidak hanya realistis tetapi juga sesuai dengan gaya atau estetika tertentu. Kemampuan ini memiliki potensi besar dalam berbagai aplikasi, mulai dari seni digital hingga pemodelan medis tiga dimensi.

2.6.3. StyleGAN2

Dibandingkan dengan pendahulunya, StyleGAN2 [53] menghadirkan perbaikan signifikan dalam hal kualitas citra yang dihasilkan serta mengatasi beberapa kekurangan yang ada pada StyleGAN awal. Salah satu keunggulan utama dari StyleGAN2 adalah kemampuannya untuk menghasilkan citra dengan detail yang lebih tajam dan tekstur yang lebih halus, tanpa adanya artefak yang biasanya muncul pada hasil generasi StyleGAN awal. Model ini memanfaatkan pendekatan berbasis gaya dengan modifikasi pada arsitektur jaringan untuk meningkatkan kualitas generasi citra. Dengan perbaikan ini, StyleGAN2 mampu menghasilkan citra yang lebih realistis dan berkualitas tinggi, menjadikannya salah satu model

generatif terdepan untuk generasi citra. Penerapan StyleGAN2 telah menunjukkan hasil yang mengesankan dalam berbagai domain, termasuk generasi citra wajah, objek, dan latar belakang yang kompleks [54]. Dalam konteks penulisan ini, penulis menggunakan model StyleGAN2 untuk mengembangkan gambar wajah yang realistis. Untuk penjelasan lebih lengkap, dapat dilihat pada bab 3 bagian "Arsitektur Model".

2.7. *Transfer Learning*

Transfer learning adalah teknik *machine learning* yang memanfaatkan model yang telah dilatih pada satu tugas sebagai titik awal untuk model pada tugas yang berbeda. Tujuan utama dari *transfer learning* adalah untuk memanfaatkan pengetahuan yang telah diperoleh dari tugas sebelumnya untuk meningkatkan pembelajaran pada tugas baru, khususnya ketika data untuk tugas baru tersebut terbatas [55][56].

Dalam konteks DL, *transfer learning* biasanya melibatkan penggunaan model yang telah dilatih pada dataset besar, seperti *ImageNet*, sebagai model dasar untuk tugas baru. Lapisan-lapisan awal dari model ini, yang telah belajar untuk mengenali fitur-fitur dasar seperti tepi dan tekstur, dapat digunakan sebagai titik awal untuk tugas baru, sementara lapisan-lapisan akhir dapat disesuaikan untuk tugas spesifik tersebut [57].

Transfer learning telah menunjukkan keberhasilan dalam berbagai aplikasi, termasuk pengenalan pola, analisis citra medis, dan pengenalan gestur berbasis elektromiografi [58][59]. Dengan memanfaatkan pengetahuan yang telah diperoleh dari tugas sebelumnya, *transfer learning* dapat mengurangi jumlah data yang diperlukan untuk melatih model yang efektif dan dapat meningkatkan kinerja model pada tugas baru.

2.8. *Fine-Tuning*

Fine-tuning adalah teknik yang digunakan dalam pembelajaran mendalam di mana model yang telah dilatih sebelumnya pada suatu tugas digunakan sebagai titik awal untuk model pada tugas yang berbeda. Tujuan utama dari *fine-tuning* adalah untuk memanfaatkan pengetahuan yang telah diperoleh dari tugas sebelumnya untuk meningkatkan pembelajaran pada tugas baru, khususnya ketika data untuk tugas baru tersebut terbatas [60][61].

Dalam konteks pembelajaran pengetuan, *fine-tuning* telah digunakan untuk menggabungkan keuntungan dari pendekatan berbasis model dan model-bebas. Misalnya, model dinamika berbasis jaringan saraf dapat digunakan bersama dengan kontrol prediktif model (MPC) untuk mencapai efisiensi sampel yang sangat baik dalam algoritma pembelajaran penguatan berbasis model. Model ini kemudian dapat disesuaikan lebih lanjut dengan metode model-bebas untuk meningkatkan kinerja pada tugas spesifik [62].

Fine-tuning juga telah digunakan dalam konteks pengenalan pola dan analisis citra. Misalnya, model GoogLeNet yang telah dilatih sebelumnya dapat disesuaikan lebih lanjut untuk klasifikasi nodul tiroid dari gambar *ultrasound*, mencapai akurasi klasifikasi yang sangat baik [60].

Namun, meskipun keberhasilan mereka, *fine-tuning* juga memiliki beberapa tantangan. Misalnya, model ini dapat mengalami masalah lupa terhadap data sebelumnya, di mana pengetahuan yang diperoleh selama pelatihan awal hilang selama *fine-tuning*. Untuk mengatasi masalah ini, beberapa penelitian telah mengeksplorasi mekanisme untuk mengingat dan belajar, yang memadukan ide-ide dari pembelajaran multi-tugas dan belajar bersama tugas pra-pelatihan dan *downstram* [63].

2.9. Pengenalan Wajah

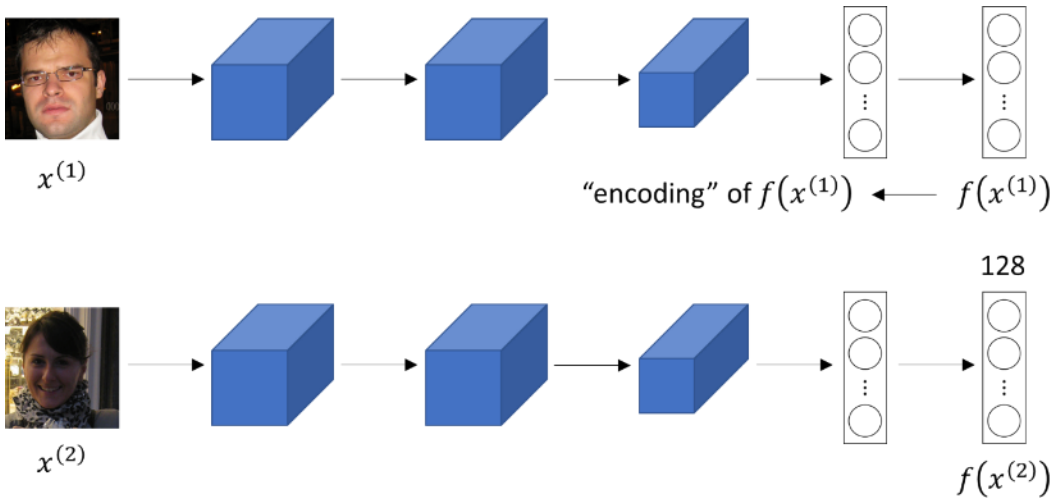
Pengenalan wajah adalah proses identifikasi dan pengenalan wajah manusia dalam gambar atau video. Teknik ini merupakan bagian penting dari banyak

aplikasi, termasuk pengawasan video. Dalam beberapa tahun terakhir, pendekatan berbasis DL telah menjadi metode pilihan untuk pengenalan wajah karena akurasi dan keandalan yang tinggi [64][65].

Pengenalan wajah menggunakan pendekatan naif seperti klasifikasi berbasis CNN mungkin tampak intuitif pada pandangan pertama, namun pendekatan ini memiliki beberapa keterbatasan yang signifikan. Pertama, pendekatan ini memerlukan pelatihan ulang model setiap kali subjek baru ditambahkan ke basis data, yang dapat menjadi sangat tidak efisien dalam skenario di mana jumlah subjek sangat besar atau berubah dengan cepat. Kedua, pendekatan ini memerlukan sejumlah besar data pelatihan untuk setiap subjek untuk mencapai kinerja yang baik, yang mungkin tidak selalu tersedia dalam aplikasi dunia nyata [5].

FaceNet adalah salah satu arsitektur DL yang paling populer untuk pengenalan wajah. Diperkenalkan oleh peneliti di Google, *FaceNet* menggunakan jaringan saraf konvolusional untuk menghasilkan *embedding* wajah, atau representasi vektor dari gambar wajah. *Embedding* ini kemudian dapat digunakan untuk mengukur kemiripan antara wajah yang berbeda. Salah satu keunggulan utama *FaceNet* adalah kemampuannya untuk menghasilkan *embedding* yang sangat diskriminatif, yang memungkinkan pengenalan wajah yang akurat bahkan dalam kondisi pencahayaan dan pose yang berbeda [66].

Sedangkan, *Siamese Network* adalah jenis arsitektur jaringan saraf yang memiliki dua atau lebih cabang identik, yaitu cabang yang melakukan operasi yang sama, memiliki parameter yang sama dan diperbarui secara bersamaan. Oleh karena itu, apa pun yang dipelajari oleh cabang juga diketahui oleh cabang lain. Gambar 2.18 merupakan ilustrasi bagaimana arsitektur *Siamese Network* bekerja.



Gambar 2. 18 Arsitektur Siamese Network

Seperti yang dalam dilihat pada gambar 2.18, arsitektur ini sangat berguna dalam kasus di mana kita peduli lebih banyak tentang bagaimana dua input berbeda daripada tentang input itu sendiri. Contoh yang umum adalah pengenalan wajah, di mana kita mungkin memiliki banyak gambar yang berbeda untuk setiap orang dan kita peduli tentang apakah dua gambar berbeda adalah gambar dari orang yang sama atau tidak.

Dalam konteks *FaceNet*, *Siamese Network* digunakan untuk menghasilkan *embedding* gambar wajah. Setiap cabang jaringan adalah CNN yang mengambil gambar wajah sebagai input dan menghasilkan *embedding* sebagai *output*. *Embedding* ini kemudian dapat digunakan untuk mengukur kemiripan antara gambar wajah.

Fungsi kerugian yang digunakan dalam *FaceNet* adalah fungsi kerugian *triplet*. Anggap kita memiliki tiga gambar: gambar *anchor*, gambar positif (gambar lain dari orang yang sama dengan *anchor*), dan gambar negatif (gambar dari orang yang berbeda), fungsi kerugian *triplet* bertujuan untuk memastikan bahwa jarak antara *anchor* dan positif lebih kecil daripada jarak antara *anchor* dan negatif oleh setidaknya margin tertentu.

Fungsi kerugian *triplet* dapat ditulis sebagai berikut:

$$L = \max(\|f(A) - f(P)\|^2 - \|f(A) - f(N)\|^2 + \alpha, 0)$$

(2. 22)

Di mana:

$f(A)$ = Gambar *anchor*.

$f(P)$ = Gambar positif.

$f(N)$ = Gambar negatif.

α = Nilai margin konstan yang ingin kita jaga antara pasangan positif dan negatif.

$\|\cdot\|$ = Norma *Euclidean*.

Berbeda dengan cara naif menggunakan arsitektur CNN tradisional, pendekatan berbasis *embedding* seperti *FaceNet* dan *Siamese Network* menawarkan solusi yang lebih fleksibel dan efisien. Dalam pendekatan ini, model dilatih untuk menghasilkan representasi vektor (*embedding*) dari gambar wajah, dan kemiripan antara dua wajah dapat diukur sebagai jarak antara *embedding* mereka. Hal ini memungkinkan model untuk mengenali subjek baru ke basis data. Selain itu, karena model ini belajar untuk membedakan antara wajah yang berbeda berdasarkan fitur yang dipelajari secara otomatis, mereka seringkali dapat mencapai kinerja yang baik bahkan dengan jumlah data pelatihan yang relatif kecil [67][68].

2.10. Fungsi Kerugian

Fungsi kerugian (*loss function*) merupakan komponen krusial dalam proses *machine learning* 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 sangat penting karena memiliki dampak langsung terhadap bagaimana model belajar dari data. Fungsi kerugian yang tidak tepat dapat menghasilkan model yang kurang optimal atau bahkan gagal belajar. Fungsi kerugian yang tepat akan memandu proses optimasi model dengan cara paling efektif, memungkinkan model untuk belajar pola yang tepat dari data dan membuat prediksi yang akurat. Berikut beberapa fungsi kerugian penting yang digunakan oleh penulis:

2.10.1. Logistic Loss

Logistic loss adalah fungsi kerugian yang biasa digunakan dalam regresi logistik dan beberapa bentuk klasifikasi biner. Fungsi kerugian ini mengukur perbedaan antara probabilitas yang diprediksi oleh model dan label sebenarnya. Dalam konteks pembelajaran mendalam, *logistic loss* dapat digunakan untuk melatih model untuk memperkirakan probabilitas bahwa suatu sampel termasuk ke kelas tertentu [69].

Dalam klasifikasi biner, *logistic loss* untuk suatu sampel dengan label y dan probabilitas yang diprediksi $\hat{y}$ didefinisikan sebagai:

$$L(y, \hat{y}) = -y \cdot \log(\hat{y}) - (1 - y) \cdot \log(1 - \hat{y})$$

(2. 23)

Di mana:

y = Label asli.

$\hat{y}$ = Label prediksi.

2.10.2. Reconstruction Loss

Reconstruction loss adalah jenis fungsi kerugian yang biasa digunakan dalam model seperti *autoencoder*, di mana tujuannya adalah untuk belajar representasi data yang dapat digunakan untuk merekontruksi input asli seakurat mungkin.

Fungsi kerugian ini mengukur perbedaan antara input asli dan versi yang direkonstruksi [70].

Untuk model seperti *autoencoder*, *reconstruction loss* biasanya diukur menggunakan *mean squared error* (MSE) antara input asli x dan versi yang direkonstruksi $\hat{x}$:

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

(2. 24)

Di mana:

n = Jumlah data.

x = Data asli.

$\hat{x}$ = Data rekonstruksi.

Meskipun penggunaan MSE sebagai metrik untuk *reconstruction loss* telah luas diadopsi, alternatif lain yang juga sering digunakan adalah L1 *loss*. Adopsi L1 *loss* dalam konteks ini bisa dijustifikasi berdasarkan beberapa alasan, salah satunya adalah penanganan terhadap *outlier* atau fluktuasi data yang ekstrem. Berbeda dengan MSE yang mengkuadratkan perbedaan antara prediksi dan target, sehingga memberikan bobot yang lebih besar pada outlier, L1 *loss* menangani perbedaan ini secara linier. Dengan demikian, L1 *loss* kurang sensitif terhadap variasi yang signifikan dalam data, yang bisa mengakibatkan nilai loss menjadi sangat besar.

Penggunaan L1 *loss* bisa menjadi strategis ketika tujuan adalah untuk mencegah *reconstruction loss* menjadi dominan dalam fungsi objektif keseluruhan, terutama dalam kasus-kasus di mana model diharapkan untuk mempertimbangkan aspek-aspek lain selain dari rekonstruksi data. Dalam situasi seperti itu, keberadaan *outlier* yang mempengaruhi MSE bisa berakibat fatal, mempengaruhi gradiasi dan

konvergensi model. Oleh karena itu, L1 *loss* bisa menjadi alternatif yang lebih kuat dalam menghitung *reconstruction loss*. L1 *loss* dapat ditulis sebagai berikut:

$$L(x, \hat{x}) = \frac{1}{n}\sum_{i=1}^{n} \|x - \hat{x}\|_1$$

(2. 25)

Di mana:

n = Jumlah data.

x = Data asli.

$\hat{x}$ = Data rekonstruksi.

2.10.3. Perceptual Loss

Perceptual loss dikenal sebagai *loss* konten, ini adalah jenis fungsi kerugian yang digunakan dalam beberapa tugas seperti *style transfer* dan super-resolusi. Fungsi kerugian ini mengukur perbedaan dalam fitur tingkat tinggi yang diekstraksi dari input dan target oleh jaringan saraf pra-latih [71].

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. 26)

Di mana:

x = Input.

y = Target.

$F(x)$ = Hasil ekstraksi input.

$F(y)$ = Hasil ektraksi target.

2.10.4. Gram Matrix Loss

Gram matrix loss adalah jenis fungsi kerugian yang digunakan dalam tugas seperti *style transfer*. Fungsi kerugian ini mengukur perbedaan antara matriks gram (yang merepresentasikan korelasi antara fitur yang diekstraksi) dari input ke target [71].

Gram matrix loss diukur menggunakan MSE antara matriks gram dari fitur yang diekstraksi dari input ke target. Misalkan $G(F(x))$ dan $G(F(y))$ adalah matriks gram dari fitur yang diesktraksi dari input x dan target y, maka *gram matrix loss* didefinisikan sebagai:

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

(2. 27)

Di mana:

x = Input.

y = Target.

$F(x)$ = Hasil ekstraksi input.

$F(y)$ = Hasil ekstraksi target.

$G(F(x))$ = Gram matriks dari hasil ekstraksi input.

$G(F(y))$ = Gram matriks dari hasil ekstraksi target.

2.10.5. Identity Preserving Loss

Identity preserving loss adalah jenis fungsi kerugian yang digunakan dalam beberapa model pengenalan wajah untuk memastikan bahwa fitur yang diekstraksi oleh model mempertahankan identitas subjek, meskipun ada variasi dalam pose, pencahayaan, ekspresi wajah, dan faktor lainnya. Dalam konteks model *ArcFace*, *identity preserving loss* digunakan bersama dengan *additive angular margin loss* (*ArcFace loss*) untuk memaksimalkan pemisahan antar kelas dan meminimalkan variasi intra-kelas [72]. *ArcFace loss* didefinisikan sebagai berikut:

$$L = -\frac{1}{N}\sum_{i=1}^{N}\log\left(\frac{e^{s\left(\cos\left(\theta_{y_i}+m\right)\right)}}{e^{s\left(\cos\left(\theta_{y_i}+m\right)\right)}+\sum_{j=1,j\neq y_i}^{n}e^{s\cos(\theta_j)}}\right)$$

(2. 28)

Di mana:

N = Jumlah sampel dalam *batch*.

s = Skala.

θ_{y_i} = Sudut antara fitur x_i dan bobot W_{y_i} untuk kelas yang benar.

m = Margin yang ditambahkan ke sudut θ_{y_i}.

Fungsi kerugian ini mendorong model untuk memperbesar sudut antara fitur dan bobot untuk kelas yang salah dan memperkecil sudut untuk kelas yang benar.

Dalam konteks *identity preserving loss*, model *ArcFace* dilatih untuk meminimalkan perbedaan antara fitur yang diekstraksi dari gambar wajah dan representasi identitas subjek yang diinginkan. Ini membantu model untuk mempertahankan identitas subjek meskipun ada variasi dalam gambar input.

2.11. Evaluasi Model

Evaluasi model adalah proses penilaian kinerja model *machine learning* atau pembelajaran mendalam. Tujuan utamanya adalah untuk memahami sejauh mana model mampu membuat prediksi yang akurat dan andal berdasarkan data yang belum pernah dilihat sebelumnya. Evaluasi model sangat penting dalam proses pengembangan model karena membantu menentukan sejauh mana model telah belajar pola yang ada dalam data dan sejauh mana model tersebut dapat digeneralisasi ke data baru.

2.11.1. Evaluasi Model Untuk Perbaikan Gambar

Evaluasi model dalam konteks perbaikan gambar atau super-resolusi seringkali melibatkan penggunaan metrik seperti LPIPS, FID, NIQE, dan PSNR. Berikut adalah penjelasan singkat tentang metrik-metrik tersebut:

1. **LPIPS (*Learned Perceptual Image Quality*):** LPIPS adalah metrik yang mengukur kesamaan *perceptual* antara dua gambar. LPIPS menggunakan jaringan saraf yang telah dilatih untuk mengukur jarak dalam ruang fitur yang lebih sesuai dengan persepsi manusia daripada jarak piksel-ke-piksel tradisional [73]. Nilai LPIPS yang lebih rendah menunjukkan kesamaan *perceptual* yang lebih tinggi antara dua gambar.
2. **FID (*Frechet Inception Distance*):** FID adalah metrik yang mengukur jarak statistik antara distribusi fitur yang diekstrak dari model *Inception v3* untuk gambar asli dan gambar yang dihasilkan. Nilai FID yang lebih rendah menunjukkan bahwa dua set gambar memiliki distribusi fitur yang lebih mirip [74]. Nilai FID yang lebih rendah menunjukkan kualitas gambar yang dihasilkan yang lebih baik. Tidak ada batas atas untuk FID, tetapi nilai minimumnya adalah 0, yang menunjukkan bahwa dua distribusi identik.
3. **NIQE (*Naturalness Image Quality Evaluator*):** NIQE adalah metrik kualitas gambar tanpa referensi yang mengukur “kealamian” gambar. NIQE menggunakan fitur statistik lokal yang diekstrak dari sub-blok

gambar dan membandingkannya dengan fitur dari gambar alami [75]. Nilai NIQE yang lebih rendah menunjukkan bahwa gambar lebih mirip dengan gambar alami.

4. **PSNR (*Peak Signal-to-Noise Ratio*):** PSNR adalah metrik yang mengukur rasio antara maksimum kemungkinan kekuatan sinyal dan kekuatan gangguan yang disebabkan oleh kesalahan. PSNR sering digunakan sebagai metriks kualitas rekonstruksi dalam masalah seperti super-resolusi gambar. Nilai PSNR yang lebih tinggi menunjukkan kualitas rekonstruksi yang lebih baik. Nilai PSNR diukur dalam dB (decibel), dengan nilai yang lebih tinggi menunjukkan kualitas yang lebih baik.

Berdasarkan penjelasan di atas, dapat dirumuskan metrik-metrik tersebut sebagai:

$$LPIPS(I_1, I_2) = \sqrt{\sum_{i=1}^{N} (f(I_1)_i - f(I_2)_i)^2}$$

(2. 29)

Di mana:

$f(I)$ = Vektor fitur yang diekstrak dari gambar *I* oleh jaringan saraf.

N = Jumlah elemen dalam vektor fitur.

$$FID = \|\mu_1 - \mu_2\|^2 + Tr\left(\Sigma_1 + \Sigma_2 - 2(\Sigma_1\Sigma_2)^{1/2}\right)$$

(2. 30)

Di mana:

$f(I)$ = Vektor fitur yang diekstrak dari gambar *I* oleh jaringan saraf.

N = Jumlah elemen dalam vektor fitur.

$$NIQE(I) = \sqrt{\sum_{i=1}^{N}(f(I)_i - f(I_{natural})_i)^2}$$

(2. 31)

Di mana:

$f(I)$ = vektor fitur yang diekstrak dari gambar *I.*

N = Jumlah elemen dalam vektor fitur.

$$PSNR(I_1, I_2) = 20\ log_{10}\left(\frac{MAX_1}{\sqrt{MSE\ (I_1, I_2)}}\right)$$

(2. 32)

Di mana:

MAX_1 = Adalah nilai maksimum yang mungkin dari gambar.

$MSE(I_1, I_2)$ = *Mean squared error* antara dua gambar.

Terdapat beberapa catatan yang perlu diketahui ketika melakukan evaluasi pada metrik-metrik di atas. Untuk mengetahui seberapa bagus evaluasi LPIPS, nilai LPIPS yang lebih rendah menunjukkan kesamaan *perceptual* yang lebih tinggi antara dua gambar.

Di sisi lain, nilai FID yang lebih rendah menunjukkan bahwa dua set gambar memiliki distribusi fitur yang lebih mirip. Nilai FID yang lebih rendah menunjukkan kualitas gambar yang dihasilkan yang lebih baik. Tidak ada batas atas untuk FID, tetapi nilai minimumnya adalah 0, yang menunjukkan bahwa dua distribusi identik.

Nilai NIQE yang lebih rendah menunjukkan bahwa gambar lebih mirip dengan gambar alami. Nilai NIQE berkisar antara 0 dan 100, dengan 0 menunjukkan gambar alami sempurna dan 100 menunjukkan perbedaan maksimal dari gambar alami.

Terakhir, nilai PSNR yang lebih tinggi menunjukkan kualitas rekonstruksi yang lebih baik. Nilai PSNR di atas 40 dB biasanya dianggap baik. Hal tersebut disebabkan oleh perbedaan antara gambar asli dan gambar yang direkonstruksi hampir sama. Tergantung pada aplikasinya, tetapi nilai PSNR sekitar 30 dB juga dapat dianggap cuku baik. Tetapi mungkin masih ada beberapa perbedaan yang terlihat antara gambar asli dan gambar yang direkonstruksi.

Perlu diketahui bagaimana cara mengevaluasi metrik-metrik di atas. Karena permasalahan yang dihadapi adalah *self-supervised learning*, metrik yang dihadapi tidak langsung ke intinya. Maksudnya adalah, evaluasi yang dilakukan pada permasalahan di atas memerlukan perbandingan dengan model lain dan permasalahan yang ingin diselesaikan.

Misalkan, untuk permasalahan *supervised learning*, peneliti dapat dengan mudah mengevaluasi model dengan *human level performance*. Tetapi, penilaian kualitas metrik evaluasi seperti PSNR, LPIPS, FID, dan NIQE sangat bergantung pada konteks aplikasi dan jenis data yang digunakan.

Dalam praktiknya, peneliti biasanya membandingkan nilai metrik evaluasi dengan nilai yang dilaporkan dalam literatur untuk metode serupa atau *benchmark* yang diakui. Jika nilai metrik sebanding atau lebih baik, model biasanya dianggap baik. Selain itu, penilaian subjektif juga sering digunakan untuk melengkapi evaluasi objektif ini